ARMY RESEARCH LABORATORY

Combined Linear and Nonlinear Radar: Waveform Generation and Capture

by Gregory J. Mazzaro and Kelly D. Sherbondy

ARL-TR-6427 **April 2013**

Approved for public release; distribution unlimited.

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Adelphi, MD 20783-1197

ARL-TR-6427 **April 2013**

Combined Linear and Nonlinear Radar: Waveform Generation and Capture

Gregory J. Mazzaro and Kelly D. Sherbondy
Sensors and Electron Devices Directorate, ARL

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. **REPORT DATE** *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED (From - To)**
April 2013	Final	December 2012–February 2013

4. TITLE AND SUBTITLE
Combined Linear and Nonlinear Radar: Waveform Generation and Capture

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Gregory J. Mazzaro and Kelly D. Sherbondy

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-SER-U
2800 P owder Mill Road
Adelphi, MD 20783-1197

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-6427

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
A combined-radar transceiver is constructed which enables target ranging in both linear and nonlinear (harmonic) receive modes. The transceiver is constructed using an arbitrary waveform generator as the signal source, a high-speed digitizing oscilloscope as the signal capture device, and commercial off-the-shelf (COTS) components for the radar front-end. A length of Subminiature Version A (SMA) cable terminated in an open circuit simulates a linear radar target; the same cable terminated in an SMA-connectorized radio front-end simulates a nonlinear radar target. A graphical user interface is developed in MATLAB to control the transceiver remotely. Ranging to the target is demonstrated experimentally using radio-frequency pulses, linear chirps, and stepped-frequency waveforms.

15. SUBJECT TERMS
Radar, linear, nonlinear, target, pulse, chirp, multitone, stepped-frequency, harmonic, transceiver

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Gregory J. Mazzaro
Unclassified	Unclassified	Unclassified	UU	66	**19b. TELEPHONE NUMBER** (*Include area code*) (301) 394-0840

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

INTENTIONALLY LEFT BLANK.

1. Introduction

In the current theater of operations, warfighters encounter threats that must be detected at a standoff distance. Some threats contain components whose permittivity contrasts substantially with that of the emplacement; such is the case with many threats that are buried. The reception of a subsurface linear radar response from an area whose surface is otherwise undisturbed indicates the presence of a threat. Others threats contain metal contacts and semiconductor junctions whose nonlinear electromagnetic response contrasts with that of the emplacement; such is the case with radio frequency (RF) electronics. The reception of a nonlinear radar response from an area that does not otherwise contain electronics indicates the presence of another class of threat. The combined radar is intended to detect both types of threats, whether or not they are collocated.

Often, threats contain dielectric, as well as electronic components, hence they will respond to both linear and nonlinear excitation. Either mode (linear/nonlinear) will detect the threat. By switching between two radar modes, additional information about the threat is received, and, thus, the probability that it is detected is improved.

1.1 Linear Radar Versus Nonlinear Radar

Linear radar is well suited to the detection of a target whose complex permittivity $\hat{\varepsilon}$ contrasts greatly with that of its surroundings:

$$\hat{\varepsilon} = \varepsilon' - j \cdot \varepsilon'', \tag{1}$$

where ε' is the "real" part, and ε'' is the "imaginary" part of the permittivity. The permittivity of a material relative to that of free space is its dielectric constant $\hat{\varepsilon}_r$:

$$\hat{\varepsilon}_r = \frac{\hat{\varepsilon}}{\varepsilon_0} = \frac{\varepsilon'}{\varepsilon_0} - j\frac{\varepsilon''}{\varepsilon_0} = \varepsilon_r' - j \cdot \varepsilon_r'', \tag{2}$$

$$\varepsilon_0 = 8.854 \cdot 10^{-12} \ \mathrm{F/m} \qquad \varepsilon_r = \left|\hat{\varepsilon}_r\right|. \tag{3}$$

Let the target be illuminated by a radar wave as illustrated in figure 1, and let the electric field of that wave E_{in} be represented by a single-tone sinusoid of frequency f_0 and amplitude E_0:

$$E_{\text{in}}(t) = E_0 \cos(2\pi \cdot f_0 \cdot t). \tag{4}$$

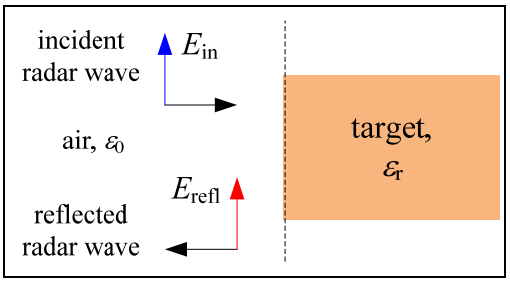

Figure 1. Incident and reflected radar waves for propagation normal to target.

Assuming normal incidence (i.e., the direction of propagation of the wave is normal to the boundary of the target), the reflected wave is (*1*)

$$E_{\text{refl}}(t) = |E_1| \cos\left(2\pi \cdot f_0 \cdot t + \phi\{E_1\}\right), \tag{5}$$

$$E_1 = \Gamma \cdot E_0 \qquad \Gamma = \frac{\sqrt{\varepsilon_{r1}} - \sqrt{\varepsilon_{r2}}}{\sqrt{\varepsilon_{r1}} + \sqrt{\varepsilon_{r2}}} = \frac{1 - \sqrt{\varepsilon_r}}{1 + \sqrt{\varepsilon_r}}. \tag{6}$$

The frequency of the reflected wave is the same as that of the incident wave, but its amplitude is scaled by the reflection coefficient Γ. For $\varepsilon_r = 1$, $\Gamma = 0$, the target is transparent to the radar wave traveling in air, and there is no reflection.

As the contrast in ε_r between a target and that of the medium through which the radar wave is propagating increases, the strength of the radar reflection from that target increases. The value of ε_r'' for a typical conductor (e.g., aluminum) is greater than 10^7 S/m (*2*). Thus, conductive targets are very detectable, even if they are buried or obscured by insulators. For insulators, ε_r'' is near zero, but ε_r' can take on a wide range of values, from $\varepsilon_r' \approx 1$ for dry foam (*3*) up to $\varepsilon_r' = 80$ for distilled water (*4*). Thus, insulating targets are not as detectable, as their radar reflections depend much more strongly on ε_r'.

Nonlinear radar exploits a completely different phenomenon: it relies on the nonlinear properties of a target to convert a portion of the incident radar wave into a reflected wave at a different frequency. Most materials found in nature are linear (with the exception of ferromagnetics), whereas many man-made materials are nonlinear (*5–11*). Semiconductor devices, such as radios and cell phones, are highly nonlinear.

A simple model for RF nonlinearity is the memoryless power series given by (*12*, *13*)

$$E_{\text{refl}}(t) = a_1 E_{\text{in}}(t) + a_2 E_{\text{in}}^2(t) + a_3 E_{\text{in}}^3(t) + \ldots = \sum_{n=1}^{N} a_n E_{\text{in}}^n(t), \tag{7}$$

where a_n are complex power-series coefficients, and E_{refl} is the electric field reflected by the target. The value of a_1 is the linear reflected response of the target, Γ; the values $\{a_2, a_3, \ldots\}$

depend upon the nonlinear properties of the target. If a nonlinear target is illuminated by the radar wave given by equation 4, the reflected wave is

$$E_{\text{refl}}(t) = \sum_{M=1}^{\infty} |E_M| \cos\left(2\pi \cdot M \cdot f_0 \cdot t + \phi\{E_M\}\right), \tag{8}$$

$$E_M = \sum_{k=1}^{\infty} \binom{2k+M-2}{k-1} \frac{a_{2k+M-2}}{2^{2k+M-3}} E_0^{2k+M-2}, \tag{9}$$

which is a sum of sinusoids at harmonics M of f_0, each with amplitude $|E_M|$ and phase $\phi\{E_M\}$. If the radar measures $E_M = 0$ for all $M > 1$, then no nonlinear target is detected. If the radar measures E_M for some $M > 1$, however, a nonlinear target is detected.

A combined radar detects targets using linear, as well as nonlinear reflective responses. The linear radar detects targets whose permittivity contrasts with that of the background, whereas the nonlinear radar detects targets whose electromagnetic properties produce a change in frequency between the incident and reflected waves.

1.2 Implementation

Linear radar can be implemented in different ways, which are commonly designated by the transmit waveform, such as continuous-wave (CW), pulsed single-tone, or chirp. To achieve an ultrawide bandwidth for ground penetration, as well as an imaging resolution, the U.S. Army Research Laboratory (ARL) designed the Synchronous Impulse Reconstruction (SIRE) radar (*14*). The SIRE radar uses a single-cycle impulse waveform, two transmit antennas, 16 receive antennas, and multiple data traces collected, whereas the radar platform is in motion in order to form high-resolution images of surface and shallow-buried targets. A single-cycle impulse and its spectrum are illustrated in figure 2a. An alternative design that allows for more flexibility in the transmitted band is the stepped-frequency waveform illustrated in figure 2b.

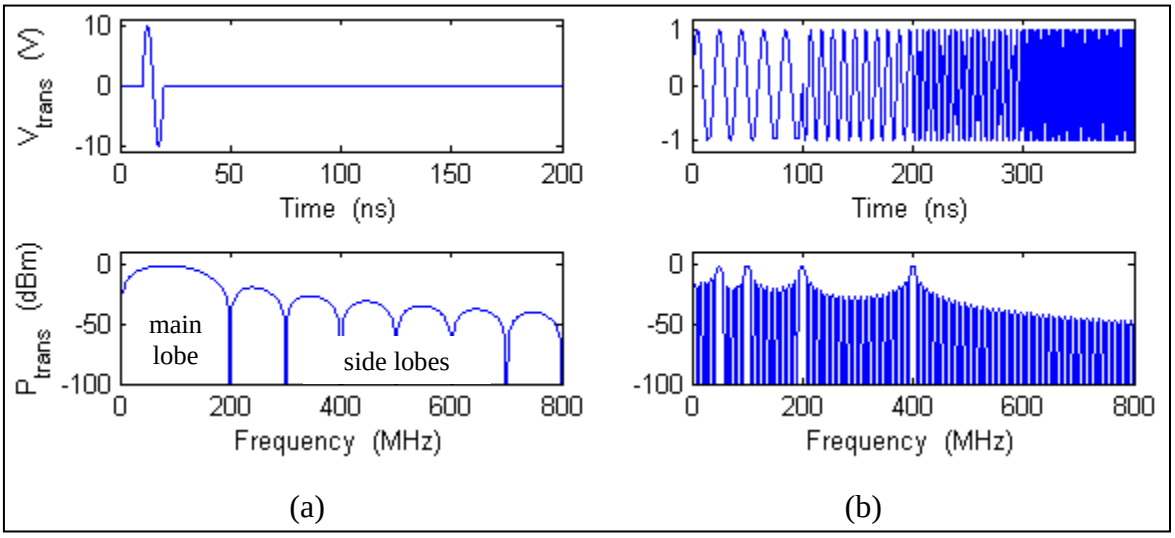

Figure 2. Linear radar waveforms: (a) impulse and (b) stepped-frequency.

Both impulse and stepped-frequency waveforms are broadband. For the impulse, the peak power is high, but the average power is low. For the stepped-frequency signal, the peak power and the average power are the same. Either waveform will provide linear detection and ranging.

One advantage of a stepped-frequency design, however, is that its underlying switched-frequency signal source is likely able to dwell on a single frequency for a long period of time. As dwell time increases while transmitting the same average power in a tone or a series of tones, the side lobes caused by interrupting the transmission (e.g., turning the source off or switching to another tone) diminish. This extended dwell time is necessary in order to minimize reflected linear side-lobes from nonlinear reflections, which are usually very weak.

Nonlinear radar can also be implemented in different ways. One popular technique is to transmit a single frequency f_0 and receive the target response at the second harmonic of the transmitted tone, $2f_0$ (*15–19*). A slight variation of this technique tracks a Doppler shift at $2f_0$ for moving targets (*20*). Other variations chirp (*21*) or digitally modulate (*12*) the transmit waveform for greater noise rejection. Another common technique is to transmit two tones f_1 and f_2 and receive the intermodulation tones $2f_1–f_2$ and $2f_2–f_1$ (*6*, *22–25*). A technique recently developed at ARL transmits at least two tones and receives not only a harmonic of the transmitted tones (e.g., $2f_1$ and $2f_2$) but also the mixing products of those tones near that harmonic (e.g., $3f_1–f_2$, $f_1 + f_2$, $3f_2–f_1$) (*26*). Figure 3a shows an example of transmit and receive spectra for a nonlinear radar that transmits one tone and receives harmonics of that tone (*27*, *28*). Figure 3b shows an example of spectra for a radar that transmits two tones and receives harmonics, as well as mixing products near those harmonics.

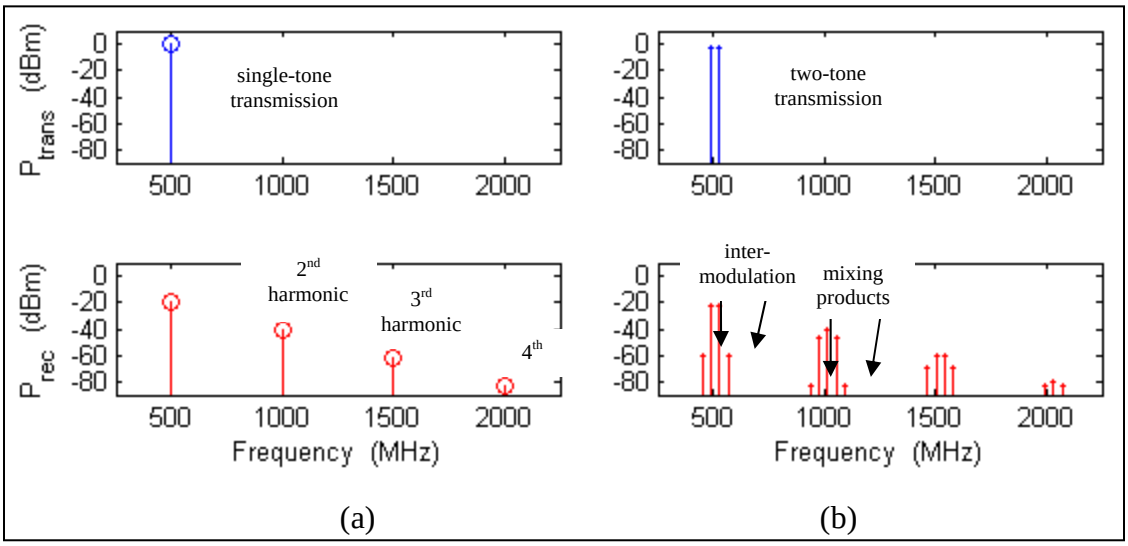

Figure 3. Nonlinear radar waveforms: (a) single-tone CW and (b) two-tone CW.

In this report, an experimental transceiver is developed to demonstrate (a) a transmit chain, which generates waveforms that are appropriate for both linear and nonlinear modes of operation, and (b) a switchable receiver chain, which captures either linear or nonlinear

responses from a radar target. The response to be exploited by the nonlinear receiver is the second harmonic of the transmitted waveform.

First, four different types of waveforms are selected that are appropriate for linear and nonlinear radar. Next, a bench-top architecture for the combined radar is proposed. Last, the experimental architecture is constructed, and measurements using a simulated (wireline) radar channel are performed to demonstrate the functionality of the linear and nonlinear radar modes.

2. Transmit Waveforms

The four waveforms selected for the linear/nonlinear transmitter are the single-tone pulse, the multitone pulse, the linear frequency-modulated (FM) chirp, and a stepped-frequency pulse.

2.1 Single-Tone Pulse

A mathematical representation for a single-tone pulse produced by an arbitrary waveform generator (AWG) is

$$V_{\mathrm{AWG}} = A_{\mathrm{env}} \cos\left(2\pi \cdot f_{\mathrm{pulse}} \cdot t\right) s\left(t\right), \tag{10}$$

with a carrier frequency f_{pulse}. The amplitude A_{env} is computed from the power of the envelope of the pulse P_{env} (in decibels referenced to 1 mW) by

$$A_{\mathrm{env}} = \sqrt{10^{P_{\mathrm{env}}^{\mathrm{dBm}}/10} \cdot 2\left(50\ \Omega\right)\left(10^{-3}\ \mathrm{W/mW}\right)}\,. \tag{11}$$

The pulse modulation is given by the switching waveform $s(t)$:

$$s\left(t\right) = u\left(t\right) - u\left(t - D_c T\right) = s\left(t + T\right) \qquad D_c T = T_{\mathrm{env}}\,, \tag{12}$$

which has a period T and a duty cycle D_{c}. The pulse is active during the time interval T_{env}. An example of an RF pulse generated by a Tektronix AWG7052 is given in figure 4. A MATLAB* function, which generates a single-tone RF pulse is given in appendix A.

It should be noted that (a) all signals in this report were captured in time by a Lecroy Wavemaster 8300A oscilloscope and in frequency by an Agilent N9342C spectrum analyzer; (b) the sampling rate of the 8300A oscilloscope was 20 GS/s, and the resolution bandwidth of the N9342C analyzer was 1 kHz; and (c) the amplitude of each waveform is less than A_{env} computed by equation 11 due to the loss introduced by the 8-ft RG-58 Subminiature Version A (SMA) cable, which feeds each of the signal capture instruments.

* MATLAB is a registered trademark of The MathWorks, Inc.

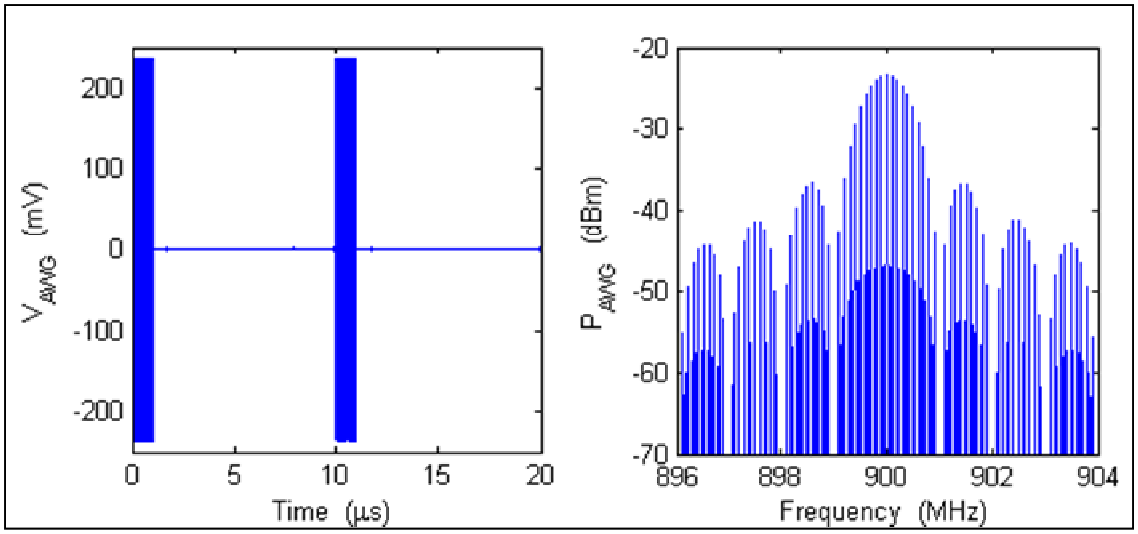

Figure 4. Single-tone RF pulse output by AWG: f_{pulse} = 900 MHz, P_{env} = 0 dBm, T_{env} = 1 μs, D_{c} = 10%.

2.2 Multitone Pulse

If, instead of a single RF carrier frequency, multiple frequencies are active during the pulse, a multitone pulse is generated:

$$V_{\text{AWG}} = A_{\text{tone}} \left\{ \cos\left(2\pi \cdot f_1 \cdot t\right) + \cos\left(2\pi \cdot f_2 \cdot t\right) + \ldots + \cos\left(2\pi \cdot f_N \cdot t\right) \right\} s\left(t\right), \tag{13}$$

which contains N frequencies given by f_1, f_2, … f_N. In this representation, the amplitude of each tone is A_{tone}, and each tone begins at a common initial phase (for maximum peak-to-average ratio, which generates a maximum nonlinear response). Also, the tones are centered at f_{c} and separated by f_{space}:

$$\frac{1}{N}\sum_{i=1}^{N} f_i = f_c \qquad f_{i+1} - f_i = f_{\text{space}}\,. \tag{14}$$

The active tones are again modulated by the on/off pulse waveform $s(t)$. An example of a multitone pulse is shown in figure 5. A MATLAB function, which generates this waveform, is given in appendix B.

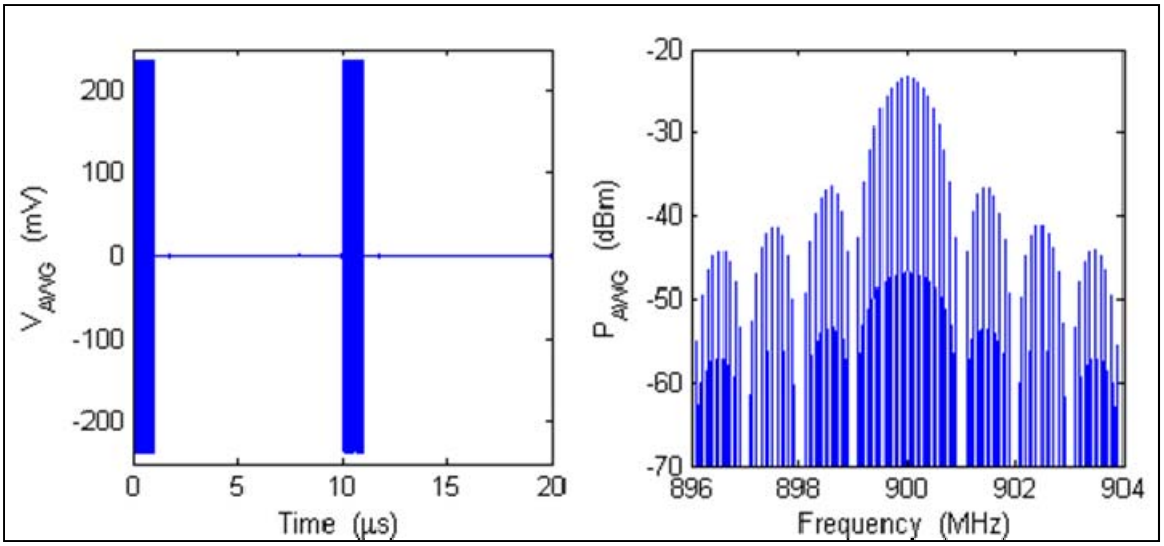

Figure 5. Multitone RF pulse output by AWG: N = 2 tones, f_c = 890 MHz, P_{tone} = –6 dBm per tone, T_{env} = 2 μs, D_c = 20%.

2.3 Linear Frequency-Modulated Chirp Pulse

A pulse whose carrier frequency begins at f_{start} and increases linearly to f_{end} over the time interval T_{env} is given by

$$V_{AWG} = A_{env} \cos\left[2\pi \cdot \left(f_{start} + (k/2)t\right) \cdot t\right] s(t) \qquad k = \left(f_{end} - f_{start}\right)/T_{env}\,, \tag{15}$$

where k is the linear chirp rate, and A_{env} is the amplitude of the pulse envelope. An example of a linear FM chirp pulse is shown in figure 6. A MATLAB function, which generates this waveform, is given in appendix C.

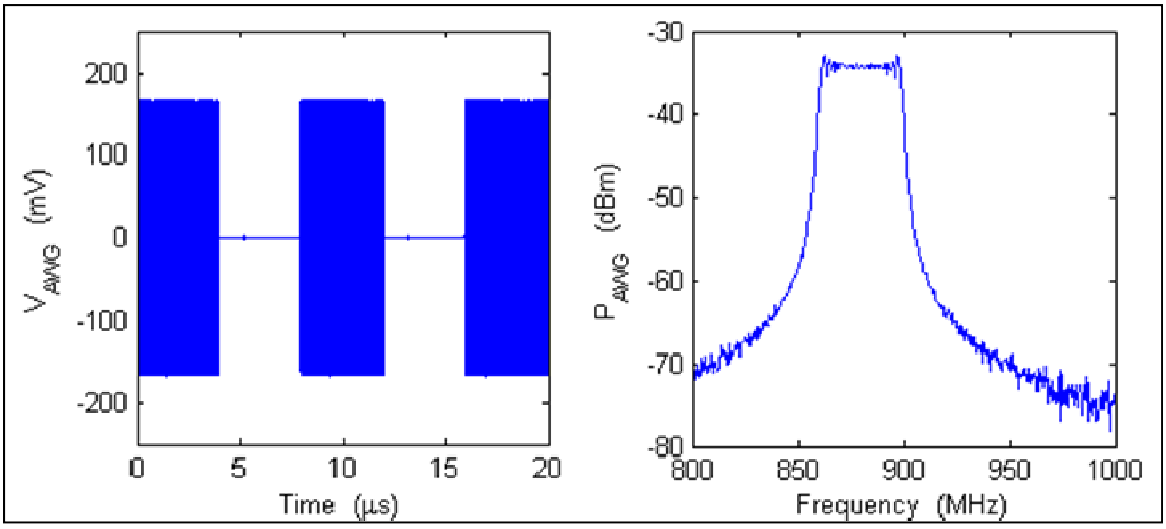

Figure 6. Linear FM chirp pulse output by AWG: f_{start} = 860 MHz, f_{end} = 900 MHz, P_{env} = –3 dBm, T_{env} = 4 μs, D_c = 50%.

2.4 Stepped-Frequency Pulse

A chirp whose carrier frequency steps between discrete values can be represented by

$$V_{\text{AWG}} = A_{\text{env}} \cos\left[2\pi \cdot f(t) \cdot t\right] s(t), \tag{16}$$

$$f(t) = \begin{cases} f_{\text{start}} & 0 \le t < \Delta t \\ f_{\text{start}} + \Delta f & \Delta t \le t < 2\Delta t \\ f_{\text{start}} + 2\Delta f & 2\Delta t \le t < 3\Delta t \\ \ldots & \ldots \\ f_{\text{end}} - \Delta f & T_{\text{env}} - \Delta t \le t < T_{\text{env}} \end{cases} \qquad \Delta t = \frac{T_{\text{env}}}{N_{\text{steps}}}, \tag{17}$$

where N_{steps} is the number of steps, T_{env} is the length of the stepped-frequency waveform, A_{env} is the amplitude, Δf is the spacing in frequency between each step, and Δt is the spacing in time between each step. It should be noted that this representation for the waveform is not phase-continuous, that is, the phase of the waveform changes abruptly across each frequency transition.

An example of a stepped-frequency pulse is shown in figure 7. A MATLAB function, which generates this waveform, is given in appendix D.

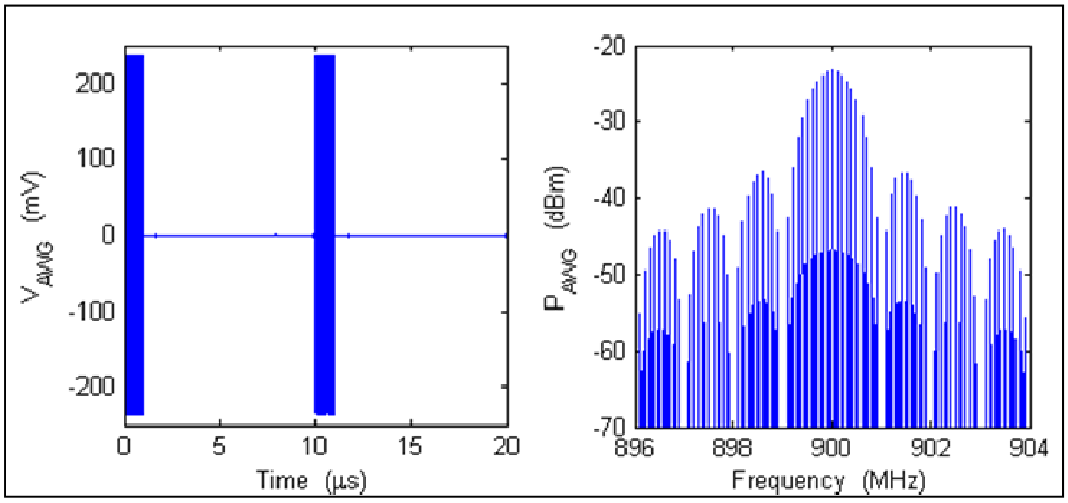

Figure 7. Stepped-frequency pulse output by AWG: f_{start} = 870 MHz, f_{end} = 890 MHz, Δf = 1 MHz, P_{env} = 0 dBm, T_{env} = 2.5 µs, D_{c} = 25%.

3. Experimental Transceiver

An architecture having components common to both linear and nonlinear modes for transmitting and receiving radar waveforms is necessary to minimize the size, weight, and power of the combined radar system. A bench-top architecture for a combined radar transceiver is given in figure 8.

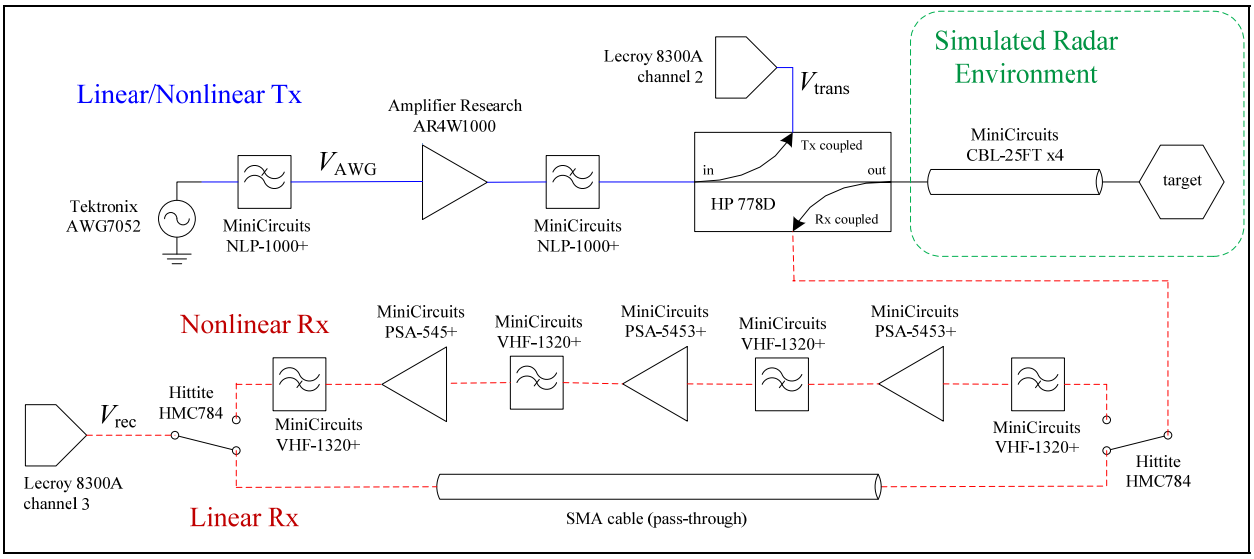

Figure 8. Experimental combined radar architecture.

In this setup, the Tektronix AWG7052 is the signal generator for both linear and nonlinear radar waveforms. The MiniCircuits NLP-1000+ low-pass filters are highly linear with a passband below 1 GHz; these filters remove much of the transmitter (Tx)-generated nonlinear (harmonic) distortion. The amplifier research AR4W1000 amplifier boosts the power of the AWG signal to a level sufficient to excite nonlinear responses from electronic targets. The HP 778D dual-directional coupler provides one port for sampling the forward (transmit, Tx) waveform and another port for sampling the reverse (receive, Rx) signal. The "Simulated Radar Environment" consists of 100 ft of SMA cable (four 25-ft cables in series), terminated by an SMA-connectorized target.

Two receive chains are selected by a pair of Hittite HMC784MS8GE switches. Each switch is powered by 5 V from the 6-V/5-A port on an Agilent E3631A supply and controlled by 5 V/0 V from the ±25-V/1-A port. In figure 8, the "Linear Rx" chain is selected, and the signal is passed directly to the 8300A oscilloscope through an SMA cable. Alternatively, the "Nonlinear Rx" chain may be selected. Along the nonlinear receiver path, the signal is filtered by three MiniCircuits VHF-1320+ high-pass filters (passbands above 1.32 GHz, to remove the linear response from capture and processing) and amplified by two MiniCircuits PSA-5453+ and one MiniCircuits PSA-545+. Each amplifier is mounted on an evaluation board and powered by 3 V from another E3631A supply.

3.1 RF Signal Generation and Capture

As measured by an Agilent N9923A network analyzer and observed in figure 9, the AR4W1000 provides more than a 40-dB gain to the transmit signal. For nonlinear (harmonic) responses, each NLP-1000+ filter attenuates Tx-generated distortion at frequencies above 1500 MHz by more than 40 dB.

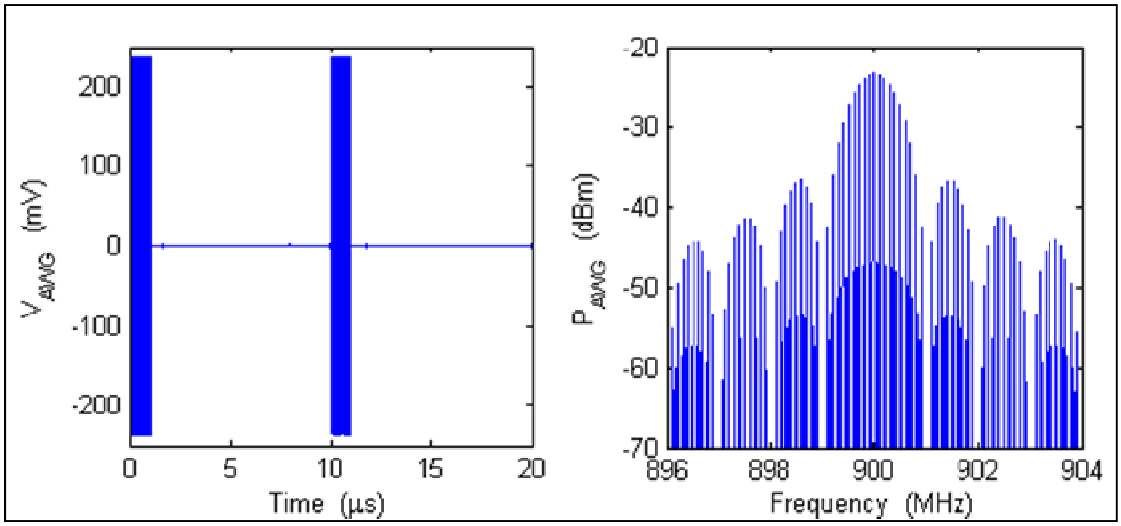

Figure 9. Transmitter amplifier and low-pass filters.

As seen in figure 10, the Tx and Rx coupling from the 778D is approximately –20 dB. Also, the nonlinear Rx chain (measured from one HMC784 "RF common" port to the other) passes signals to the 8300A with a gain of approximately 40 dB, whereas the linear Rx chain passes signals through with a loss under 3 dB.

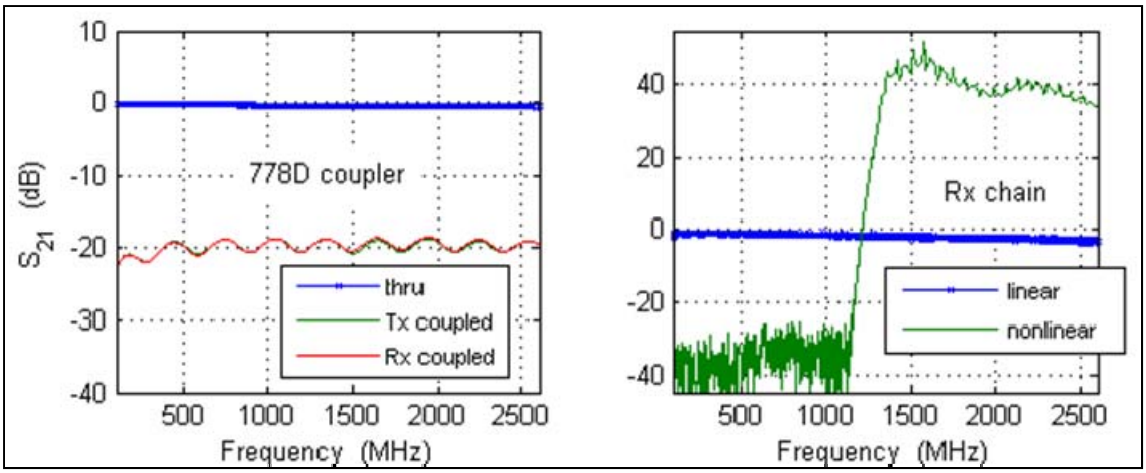

Figure 10. Directional coupler and linear/nonlinear receiver chain.

3.2 MATLAB Graphical User Interface

The AWG7052 generator, 8300A oscilloscope, and E3631A supplies are controlled via the General Purpose Interface Bus (GPIB). Communication is established using the Instrument Control Toolbox in MATLAB (v7.0.0.19920, R14). The graphical user interface (GUI) in figure 11 was created using MATLAB's "guide" function. The script and functions that govern the operation of the GUI are given in appendices E through K.

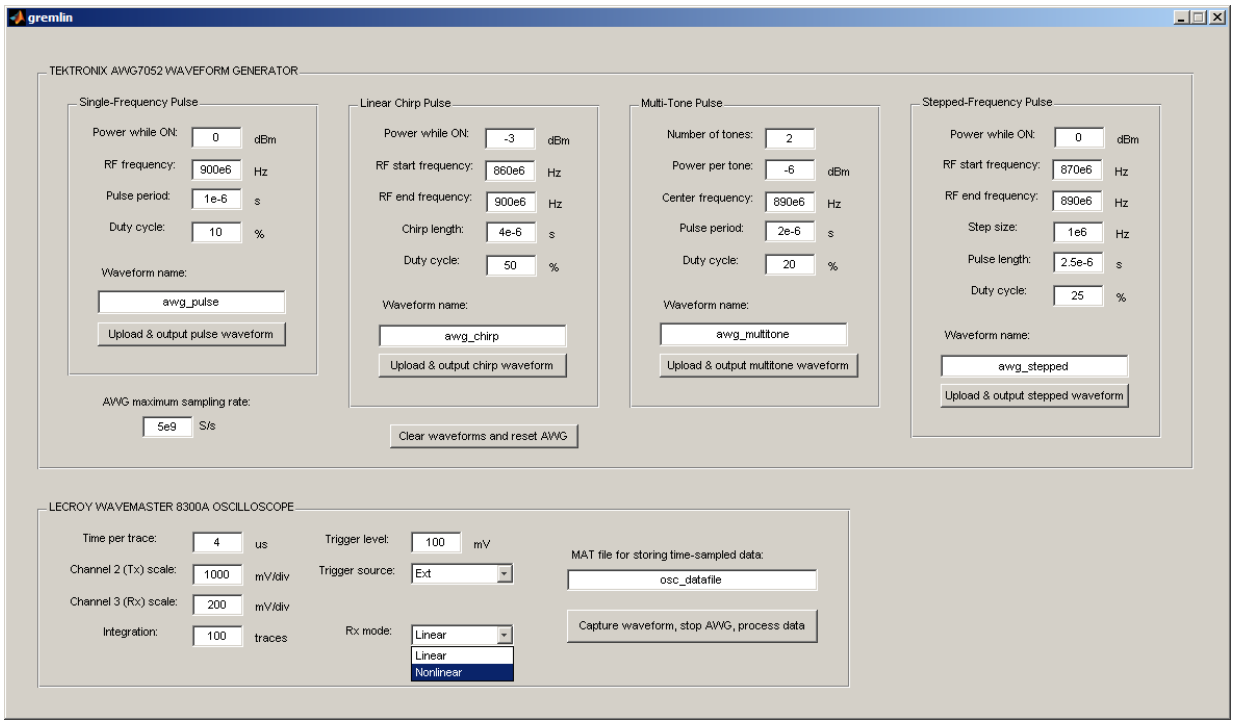

Figure 11. GUI for experimental combined-radar system.

Using the upper panel, the four different waveforms presented in section 2 may be uploaded to the AWG for transmission to the target. Using the lower panel, the signal from the target may be captured using the linear or nonlinear receive chain and processed accordingly.

For the single-tone pulse, the user may choose the power of the RF pulse while it is active (P_{env}), the RF frequency (f_{pulse}), the time interval during which the pulse is active (T_{env}), and the pulse duty cycle (D_c). The "waveform name" is the designation that appears on the AWG after the waveform is uploaded and is selected for waveform playback.

For the linear FM chirp pulse, the user may choose the power of the chirp envelope (P_{env}), the frequency at which the chirp starts (f_{start}), the frequency at which the chirp ends (f_{end}), the time interval during which the frequency linearly changes from f_{start} to f_{end} (T_{env}), and the duty cycle of the waveform (D_c).

For the multitone pulse, the user may choose the number of tones (N), power per tone (P_{tone}), the frequency at which the tones are centered (f_c), the time interval during which the pulse is active (T_{env}), and the pulse duty cycle (D_c). The frequency separation between the tones (f_{space}) is automatically set to $1/T_{env}$, so that the shortest waveform necessary to achieve N, f_c, and T_{env} with negligible frequency aliasing is uploaded to the AWG.

For the stepped-frequency waveform, the user may choose the power of the pulse envelope (P_{env}), the frequency at which the stepping starts (f_{start}), the frequency at which the stepping ends

(f_{end}), the time interval during which the frequency steps from f_{start} to f_{end} (T_{env}), the step size (Δf), and the duty cycle of the waveform (D_c).

In agreement with figure 8, the sampled Tx signal is fed to channel 2 of the 8300A oscilloscope, and the sampled Rx signal is fed (through the linear/nonlinear receive chain) to channel 3. The user chooses the voltage scale per channel, the total data collection time per trace, and the number of integrations (i.e., the number of data traces averaged before capture).

The user chooses the trigger level and source. In the experimental setup, Marker 1 from the AWG7052 is fed to the External trigger port on the 8300A.

The user chooses the receiver (Rx) mode and types a name for the native MATLAB (MAT) file that will store the time-sampled Tx and Rx voltage vectors.

Upon pressing the "Upload…" button inside of one of the upper subpanels, the appropriate waveform is generated and sent to the AWG. A new figure panel (not shown) appears, which plots the software-generated waveform in frequency and time to confirm that the signal the user intended has been uploaded.

Upon pressing the "Capture…" button inside the lower subpanel, the corresponding signal received from the target is recorded by the oscilloscope and processed in MATLAB. A second figure panel (shown in section 4) appears, which plots the raw Tx and Rx data in time. A third figure panel (also shown in section 4) appears, which plots the correlation of the Tx and Rx voltage samples.

---

4. Wireline Experiments

Several experiments were conducted in order to demonstrate the performance of the bench-top combined-radar transceiver: three different waveforms (pulse, linear chirp, stepped-frequency), two Rx modes (linear and nonlinear), two linear targets (open-circuit, matched load), and one nonlinear target (Family Radio Service [FRS] radio).

4.1 Linear Rx, Chirp Waveform, Open-Circuit Target

Figure 12 shows the result of the linear data capture and processing when reflecting a chirp from an open-circuit target. Figure 12a plots the raw Tx and Rx data. Figure 12b plots the cross correlation of the Tx and Rx signals:

$$\left[V_{\text{trans}} * V_{\text{rec}}\right](t) = \int_{-\infty}^{+\infty} V_{\text{trans}}(t) \cdot V_{\text{rec}}(t+\tau)\, d\tau\,, \tag{18}$$

where time has been mapped to distance using the velocity of propagation of an RF signal in the MiniCircuits CBL 25-ft coaxial lines (dielectric constant $\varepsilon_r \approx 2.1$):

$$d = \frac{1}{2} \cdot t \cdot \frac{c}{\sqrt{\varepsilon_r}} = \left(\frac{t}{2}\right)\left(\frac{3 \cdot 10^8 \text{ m/s}}{\sqrt{2.1}}\right)\left(\frac{3.28 \text{ ft}}{1 \text{ m}}\right) = \left(0.34 \frac{\text{ft}}{\text{ns}}\right) t \,. \tag{19}$$

A factor of 1/2 is used in equation 19, because the distance plotted is half the round-trip distance from the transmitter (i.e., from the coupler output port) to the target (i.e., to the end of the 100-ft coaxial line) to the receiver (i.e., back to the coupler output port).

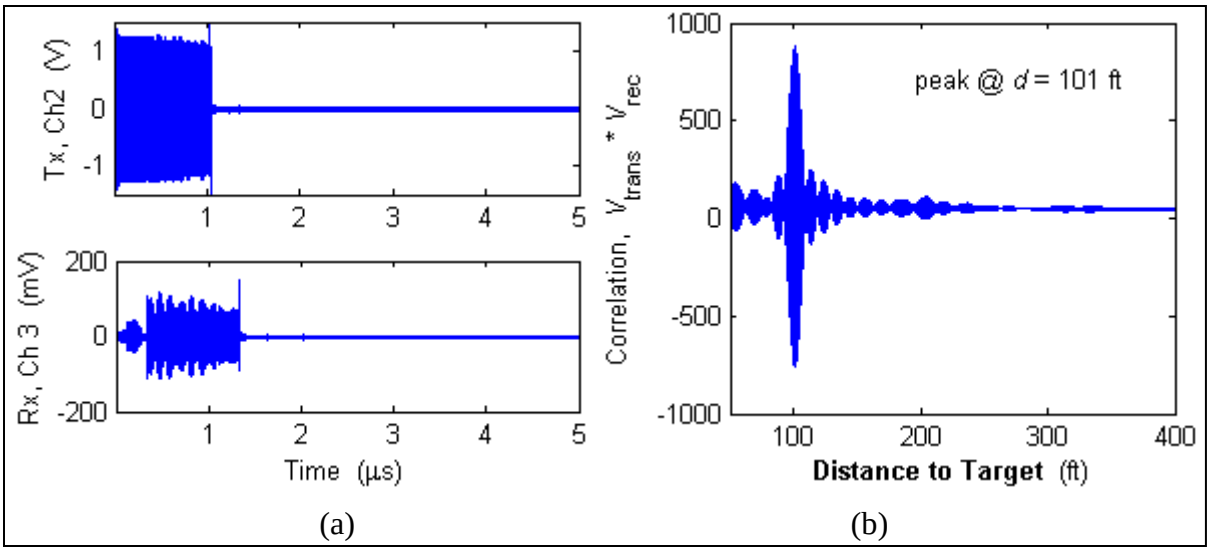

Figure 12. Radar data, chirp Tx waveform, linear Rx mode, open-circuit target: f_{start} = 880 MHz, f_{end} = 920 MHz, P_{env} = 0 dBm, T_{env} = 1 µs, D_c = 10%, (a) raw Tx and Rx data and (b) correlation of Tx and Rx waveforms.

Cross correlation is a basic form of target ranging. The peak of V_{trans}*V_{rec} (as a function of distance) indicates the distance from the transmitter to the target.

In figure 12a, a relatively constant-amplitude pulse is visible in the sampled Tx channel, and a distorted pulse is visible in the sampled Rx channel. These waveforms are expected, given the frequency-dependent characteristic of the coupler in figure 10. In figure 12b, a sinc function is visible along with several sidelobes. This shape is expected from the cross correlation of two chirps. The peak of the sinc function is visible at a distance of d = 101 ft. The calculated distance is very close to the length of the coaxial line (and slightly higher because the calculation does not account for the length of the Rx chain).

4.2 Linear Rx, Chirp Waveform, Open Circuit Versus Matched Load

Figure 13a shows the result of the cross correlation when reflecting a chirp with a wider bandwidth than that of section 4.1 from an open-circuit target. Figure 13b is the correlation when reflecting the same waveform from a matched (50-Ω) load. Two results are notable: (1) the peak is sharper when the bandwidth of the Tx waveform is wider and (2) very little signal reflects from the matched load. Both results are expected and indicate proper operation of the transmitter and the linear receive chain.

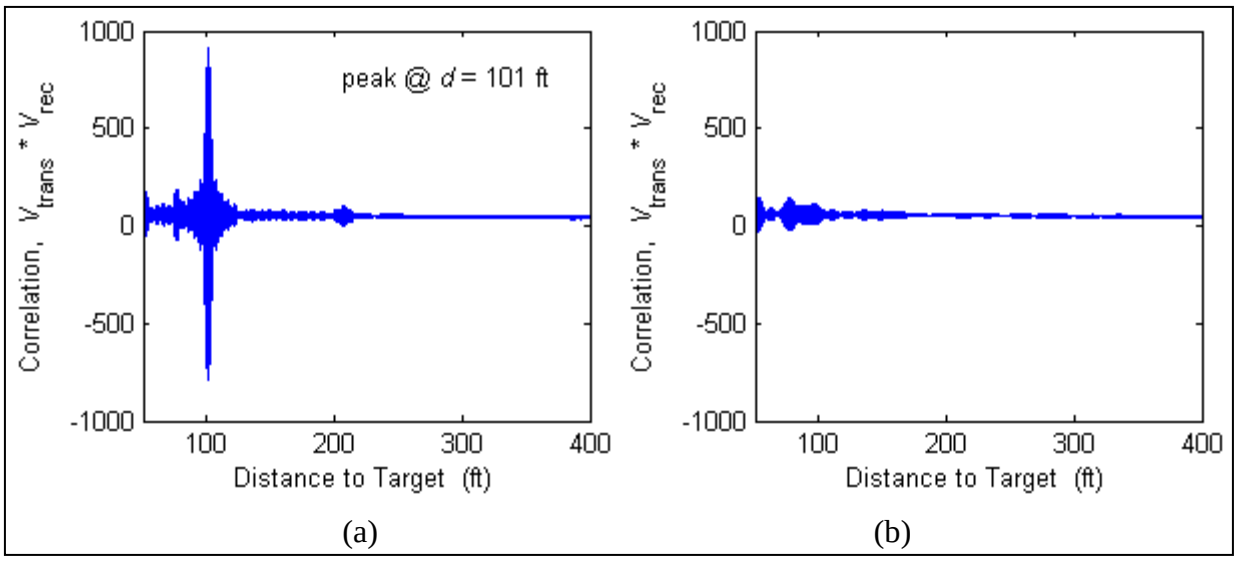

Figure 13. Radar data, chirp Tx waveform, linear Rx mode: f_{start} = 860 MHz, f_{end} = 940 MHz, P_{env} = 0 dBm, T_{env} = 1 μs, D_c = 10%, (a) open-circuit target, (b) matched-load target.

4.3 Nonlinear Rx, Pulse Waveform, Nonlinear Target

Figure 14 shows the result of the nonlinear data capture when reflecting an RF pulse from a nonlinear target: a Motorola T4500 radio whose antenna has been replaced by an SMA end-launch connector. A 13-dB attenuator is placed between the end of the coaxial line and the FRS radio. Figure 14a plots the raw Tx and Rx data along a 5-μs time scale. Figure 14b plots the same raw data along a 2-ns time scale between t = 500 ns and t = 502 ns.

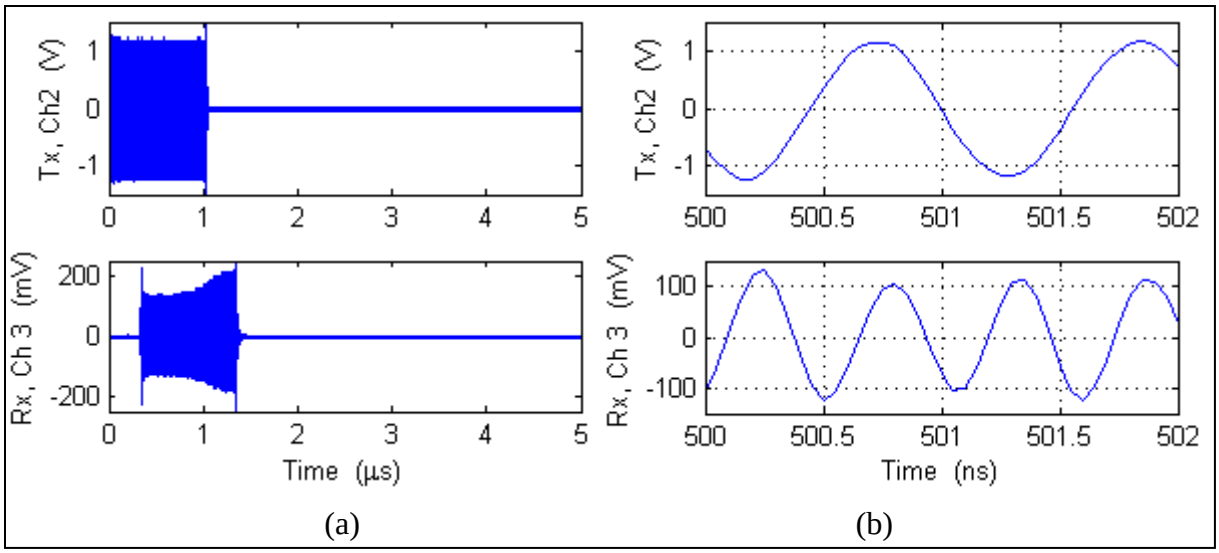

Figure 14. Radar data, RF pulse Tx waveform, linear Rx mode, FRS radio target: f_{pulse} = 900 MHz, P_{env} = 0 dBm, T_{env} = 1 μs, D_c = 10%, (a) raw data, complete time scale and (b) raw data, zoomed-in time scale.

It is not obvious from figure 14a that the transceiver is detecting the nonlinear response from the target. In figure 14b, however, the observed response is clearly nonlinear, because the frequency of the received signal (1800 MHz) is twice that of the transmitted signal (900 MHz).

4.4 Nonlinear Rx, Stepped-Frequency Waveform, Nonlinear Target

Nonlinearity is also visible in the frequency domain when the Tx and Rx signals are captured with a spectrum analyzer. Figure 15 provides such captures for a stepped-frequency waveform and the FRS radio target. The signal output from the AWG (and filtered by a NLP-1000+) is plotted above, and the received spectrum is plotted below. For P_{AWG}, all of the spectral content is centered at f = 900 MHz, and no spectral content exists near $2f$ = 1800 MHz. For P_{rec}, all of the spectral content is centered at $2f$ = 1800 MHz, and no spectral content exists near f = 900 MHz.

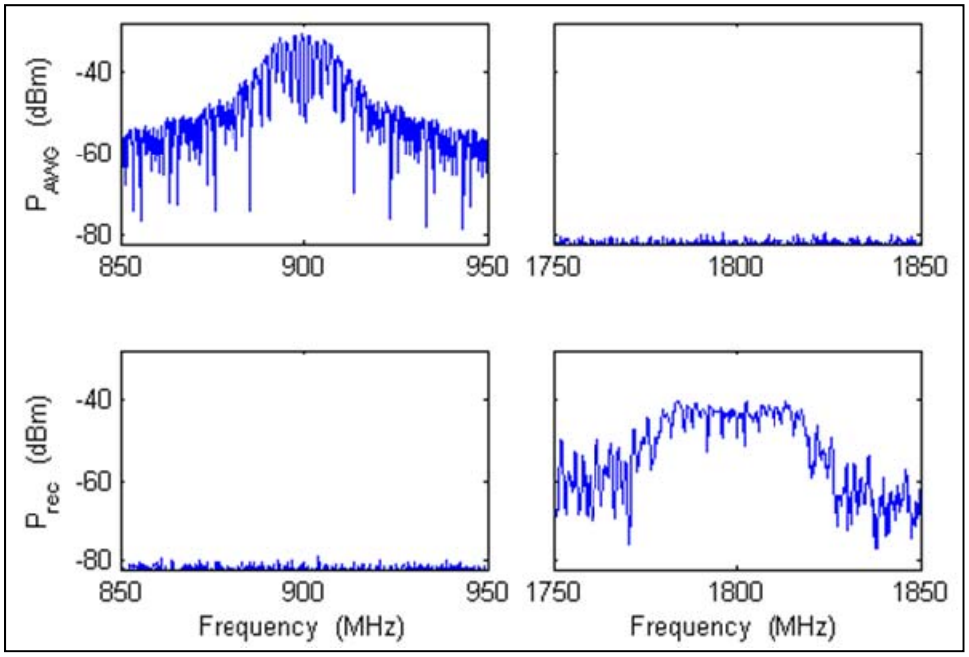

Figure 15. Tx and Rx frequency content, stepped-frequency Tx waveform, nonlinear Rx mode, FRS radio target: f_{start} = 890 MHz, f_{end} = 910 MHz, Δf = 1 MHz, P_{env} = 0 dBm, T_{env} = 2 μs, D_{c} = 20%.

4.5 Nonlinear Rx, Chirp Waveform, Nonlinear Target

Figure 16 shows the result of the nonlinear data capture and processing when reflecting a chirp from the FRS radio. Figure 16a plots the raw Tx and Rx data. Figure 16b plots the cross correlation of the Tx and Rx signals:

$$\left[V_{\mathrm{trans}} * V_{\mathrm{rec}}\right](t) = \int_{-\infty}^{+\infty} V'_{\mathrm{trans}}(t) \cdot V_{\mathrm{rec}}(t+\tau)\, d\tau\,, \tag{20}$$

where the Tx signal used for the correlation is a filtered second harmonic of the captured V_{trans}:

$$V'_{\text{trans}}(t) = h_{\text{BPF}}(t) * V^2_{\text{trans}}(t), \tag{21}$$

and h_{BPF} is a bandpass filter with passband edges $f_L = 3f_c/2$ and $f_U = 5f_c/2$, where $f_c = (f_{\text{start}} + f_{\text{end}})/2$.

A sinc function is again visible and centered at d = 103 ft. This distance is longer than d = 101 ft measured previously, because the nonlinear Rx chain contains slightly more propagation delay (through the filters and amplifiers) than the linear Rx chain (SMA cable, pass through).

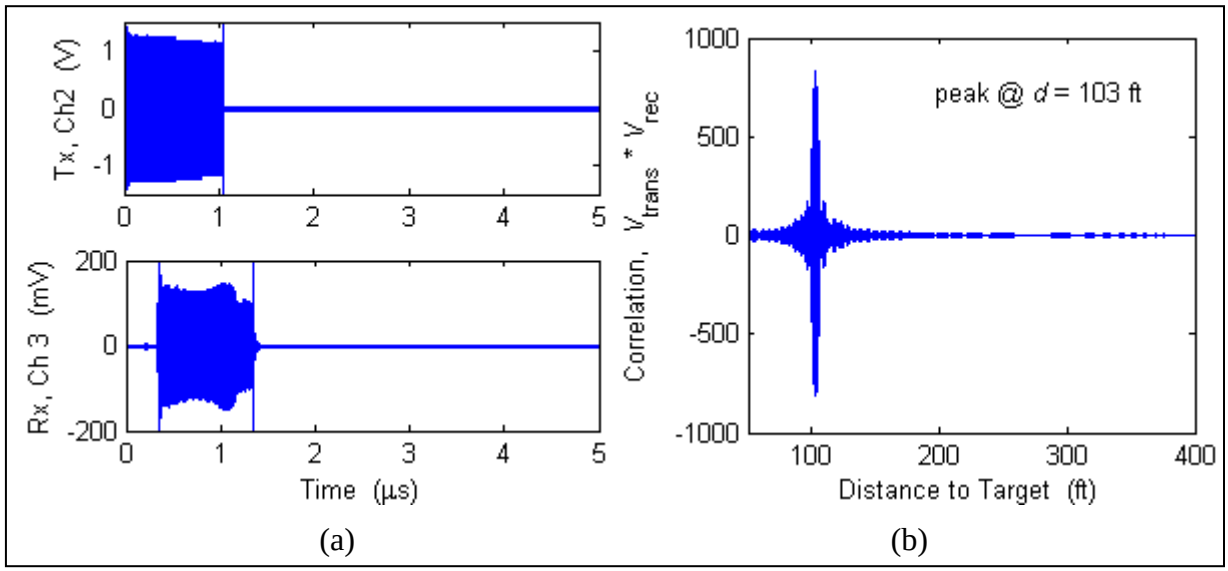

Figure 16. Radar data, linear chirp Tx waveform, nonlinear Rx mode, FRS radio target: f_{start} = 880 MHz, f_{end} = 920 MHz, P_{env} = 0 dBm, T_{env} = 1 μs, D_c = 10%, (a) raw Tx and Rx data and (b) correlation of Tx and Rx waveforms.

4.6 Nonlinear Rx, Chirp Waveform, Nonlinear Versus Open-Circuit Target

The nonlinear Rx chain was tested against a purely linear target in order to demonstrate that the transceiver does not indicate detection if the target is linear, and the radar is listening for a nonlinear response. Figure 17 gives the result of this test, which is performed with a chirp waveform.

From figure 17a, it is clear that the radar registers a detection (d = 103 ft) when the target is nonlinear, and the Rx is expecting a nonlinear response. From figure 17b, it is clear that the nonlinear Rx chain does not register a detection when the target is linear.

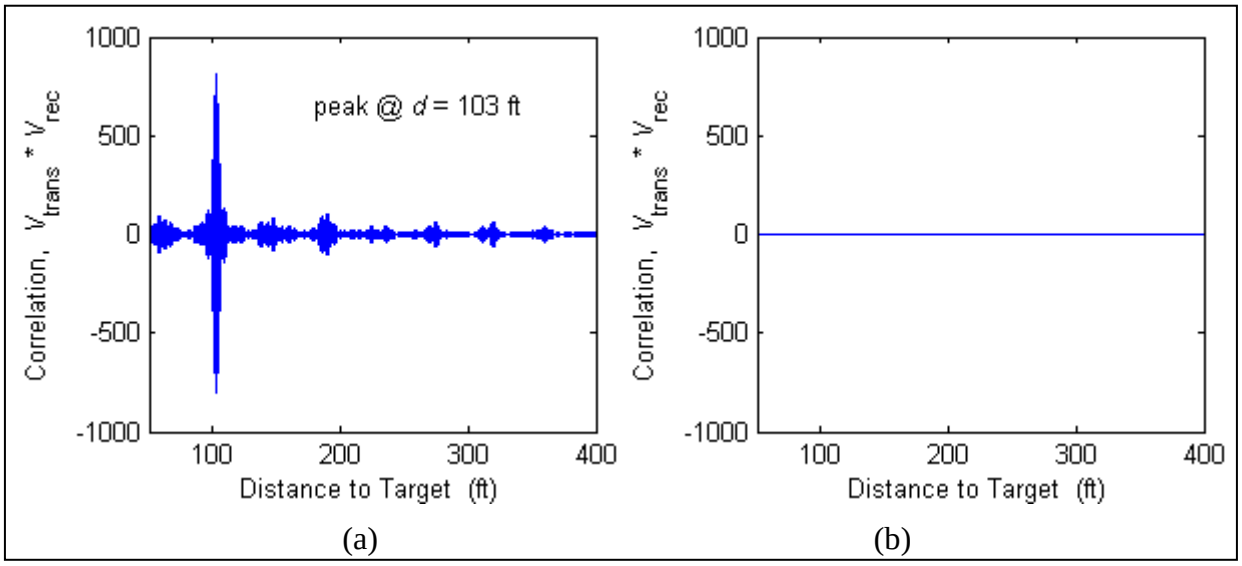

Figure 17. Radar data, chirp Tx waveform, nonlinear Rx mode: f_{start} = 890 MHz, f_{end} = 910 MHz, P_{env} = 0 dBm, T_{env} = 1 μs, D_c = 10%, (a) FRS radio target and (b) open-circuit target.

5. Conclusions

A combined-radar transceiver has been constructed, which enables basic target ranging in both linear and nonlinear (harmonic) receive modes. The transceiver was constructed using an arbitrary waveform generator as the signal source, a high-speed digitizing oscilloscope as the signal capture device, and commercial off-the-shelf (COTS) components for the radar front-end (amplification, filtering, and switching). A 100-ft length of SMA cable terminated in an open circuit simulated a linear radar target; the same cable terminated in an SMA-connectorized FRS radio simulated a nonlinear radar target. A MATLAB GUI was developed in order to control the transceiver remotely. The associated script and helper functions are provided in the appendices. Ranging to the target was demonstrated experimentally using RF pulses, linear FM chirps, and stepped-frequency waveforms.

6. References

1. Ulaby, F. T. *Fundamentals of Applied Electromagnetics*; media ed. Upper Saddle River, NJ, USA: Pearson Education, Inc., 2004.

2. Ramo, S.; Whinnery, J. R.; Van Duzer, Van *Fields and Waves in Communication Electronics*; 3rd ed. Hoboken, NJ, USA: John Wiley & Sons, Inc., 1994.

3. Waldron, I.; Makarov, S. N. Measurement of Dielectric Permittivity and Loss Tangent for Bulk Foam Samples with Ssuspended Ring Resonator Method. in *Proc. IEEE Ant. Prop. Soc. Int. Symp.*, pp 3175–3178, July 2006.

4. Drake, F. H.; Pierce, G. W.; Dow, M. T. Measurement of the Dielectric Constant and Index of Refraction of Water and Aqueous Solutions of KCl at High Frequencies. *Phys. Rev.* **Mar. 1930**, *35* (6), 613–622.

5. Arazm, F.; Benson, A. Nonlinearities in Metal Contacts at Microwave Frequencies. *IEEE Trans. Electromagn. Compat.* **Aug. 1980**, *22* (3), 142–149.

6. Ehlers, G. L.; Charters, J. P. Semiconductor Article Harmonic Identification, U.S. Patent 6,856,275, Feb. 15, 2005.

7. Wilkerson, J. R.; Lam, P. G.; Gard, K. G., Steer, M. B. Distributed Passive Intermodulation Distortion on Transmission Lines. *IEEE Trans. Microw. Theory Tech.* **May 2011**, *59* (5), p. 1190–1205, May 2011.

8. Lui, P. L.; Rawlins, A. D. Passive Non-linearities in AntennaSystems. in *Proc. IEE Colloq. Passive Intermod. Prod. Ant. Related Struct.*, pp. 1–7, June 1989.

9. Mazzaro, G. J.; Steer, M. B.; Gard, K. G. Intermodulation Distortion in Narrowband Amplifier Circuits. *IET Microw. Antennas Propagat.* **Sept 2010**, *4* (9), 1149–1156.

10 Henrie, J.; Christianson, A.; Chappell, W. J. Prediction of Passive Intermodulation from Coaxial Connectors in Microwave Networks. *IEEE Trans. Microw. Theory Tech.* **Jan 2008**, *56* (1), 209–216.

11. Bailey, G. C.; Ehrlich, A. C. A Study of RF Nonlinearities in Nickel. *J. Appl. Phys.* **Jan 1979**, *50* (1), 453–461.

12. Polacek, V.; Pavlik, R. The Use of Digital Modulation Signals in Radar System for Detection of Nonlinear Scatterers. in *Proc. Int. Radar Symp., IRS*, pp. 743–747, Sept. 2011.

13. Crowne, F.; Fazi, C. Nonlinear Radar Signatures from Metal Surfaces. in *Proc. Int. Radar Conf., RADAR*, pp. 1–6, Oct. 2009.

14. Koenig, F.; Ressler, M.; Smith, G.; Nguyen, L.; Harris, R. *Synchronous Impulse Reconstruction (SIRE) Radar Sensor*; ARL-TR-4661; U.S. Army Research Laboratory: Adelphi, MD, Nov. 2008.

15. Brazee, R. D., et al. A Transponder for Harmonic Radar Tracking of the Black Vine Weevil in Behavioral Research. *Trans. Amer. Soc. Agricult. Eng.* **2005**, *48* (2), 831–838.

16. Colpitts, B. G.; Boiteau, G. Harmonic Radar Transceiver Design: Miniature Tags for Insect Tracking. *IEEE Trans. Antennas Propag.* **Nov 2004**, *52* (11), 2825–2832.

17. O'Neal, M. E.; Landis, D. A.; Rothwell, E.; Kempel, L.; Reinhard, D. Tracking Insects with Harmonic Radar: A Case Study. *Amer. Ento.* **2004**, *50* (4), 212–218.

18. Psychoudakis, D.; Moulder, W.; Chen, C. C.; Zhu, H.; Volakis, J. L. A Portable Low-power Harmonic Radar System and Conformal Tag for Insect Tracking. *IEEE Antennas Wireless Propag. Lett.* **2008**, *7*, 444–447.

19. Tahir, N.; Brooker, G. Recent Developments and Recommendations for Improving Harmonic Radar Tracking Systems. in *Proc. 5th European Conf. Ant. Propagat., EUCAP*, pp. 1531–1535, Apr. 2011.

20. Singh, A.; Lubecke, V. Respiratory Monitoring and Clutter Rejection Using a CW Doppler Radar with Passive RF Tags. *IEEE Sensors* **Mar 2012**, *12* (3), 558–565.

21. Stagner, C.; Conrad, A.; Osterwise, C.; Beetner, D. G.; Grant, S. A Practical Superheterodyne-receiver Detector Using Stimulated Emissions. *IEEE Trans. Instrum. Meas.* **Apr. 2011**, *60* (4), 1461–1468.

22. Saebboe, J. et al. Harmonic AutomotiveRadar for VRU classification. in *Proc. Int. Radar Conf., RADAR*, pp. 1–5, Oct. 2009.

23. Martone, A. F.; Delp, E. J. Characterization of RF Devices Using Two-tone Probe Signals. in *Proc. 14th Workshop on Stat. Sig. Process., IEEE/SP*, pp. 161–165, Aug. 2007.

24. Lyons, R. G. Signal and Interference Output of a Bandpass Nonlinearity. *IEEE Trans. Commun.* **June 1979**, *27* (6), 888–891.

25. Walker, A. L.; Buff, P. M. Method and Apparatus for Remote Detection of Radio-frequency Devices. U.S. Patent 8,131,239, Mar. 6, 2012.

26. Mazzaro, G. J.; Martone, A. F. Detection of RF Electronics by Multitone HarmonicRadar. *journal article in preparation*, Oct. 2012.

27. Lehtola, G. E. RF Receiver Sensing by Harmonic Generation. U.S. Patent 7,864,107, Jan. 4, 2011.

28. Holmes, S. J.; Stephen, A. B. Non-linear Junction Detector. U.S. Patent 6,897,777, May 24, 2005.

Appendix A. MATLAB Function for Generating Single-Tone RF Pulses†

† This appendix is presented in its original form without editorial change.

```
function [v, f_sample] = awg_pulse( f_pulse, T_env, P_env, duty_cycle, f_sample_max )

f_res = 1/T_env;                            % frequency resolution (Hz)

f_sample = 2*f_res;
while (f_sample <= f_sample_max/2)          % adjust AWG sample rate
    f_sample = 2*f_sample;
end

f_pulse = round(f_pulse/f_res)*f_res;       % adjust pulse carrier frequency

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

t_sbb = 1/f_sample;                         % sampling time
t = 0:t_sbb:T_env-t_sbb;                    % time vector
A = sqrt(10^(P_env/10)*2*50*10^-3);         % amplitude of pulse when ON

x = A * cos(2*pi*f_pulse.*t);               % sinusoid equation

if (duty_cycle <= 0) || (duty_cycle >= 1)
    v = x;                                  % continuous wave
else
    v = [x zeros(1,(round(1/duty_cycle)-1)*length(x))];     % pulsed
end
```

Appendix B. MATLAB Function for Generating Multitone RF Pulses‡

‡ This appendix is presented in its original form without editorial change.

```
function [v, f_sample] = awg_multitone( N, f_c, T_env, P_tone, duty_cycle, f_sample_max )

f_res = 1/T_env;                            % resolution based on envelope length
f_space = 2*f_res;                          % tone spacing

f_sample = 2*f_res;
while (f_sample <= f_sample_max/2)          % adjust AWG sample rate
    f_sample = 2*f_sample;
end

f_c = round(f_c/f_res)*f_res;               % adjust center frequency

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

A = sqrt(10^(P_tone/10)*2*50*10^-3);        % amplitude of each tone

t_sbb = 1/f_sample;                         % sampling time
t = 0:t_sbb:T_env-t_sbb;                    % time vector

xi = zeros(1,length(t));                    % initialize baseband I/Q vectors
xq = zeros(1,length(t));

% generate multitones with in-phase and quadrature components
for n = 1:N/2
    xi = xi + (A)*cos(2*pi*f_space*(abs(n)-1/2)*t) + (A)*cos(2*pi*f_space*(abs(n)-1/2)*t);
    xq = xq + (A)*sin(2*pi*f_space*(abs(n)-1/2)*t) - (A)*sin(2*pi*f_space*(abs(n)-1/2)*t);
end;

x = xi.*cos(2*pi*f_c*t ) - xq.*sin(2*pi*f_c*t );

if (duty_cycle <= 0) || (duty_cycle >= 1)
    v = x;                                  % continuous wave
else
    v = [x zeros(1,(round(1/duty_cycle)-1)*length(x))];     % pulsed
end
```

Appendix C. MATLAB Function for Generating RF Chirp Pulses[§]

[§] This appendix is presented in its original form without editorial change.

```
function [v, f_sample] = awg_chirp( f_start, f_end, T_env, P_env, duty_cycle, f_sample_max )

f_res = 1/T_env;                            % frequency resolution (Hz)

f_sample = 2*f_res;
while (f_sample <= f_sample_max/2)          % adjust AWG sample rate
    f_sample = 2*f_sample;
end

f_start = round(f_start/f_res)*f_res;       % adjust start frequency
f_end = round(f_end/f_res)*f_res;           % adjust end frequency

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

t_sbb = 1/f_sample;                         % sampling time
t = 0:t_sbb:T_env-t_sbb;                    % time vector
A = sqrt(10^(P_env/10)*2*50*10^-3);         % amplitude of chirp when ON

k = (f_end-f_start)/T_env;                  % chirp rate (Hz/s)

x = A * cos(2*pi*(f_start+(k/2)*t).*t);     % chirp equation

if (duty_cycle <= 0) || (duty_cycle >= 1)
    v = x;                                  % continuous wave
else
    v = [x zeros(1,(round(1/duty_cycle)-1)*length(x))];     % pulsed
end
```

Appendix D. MATLAB Function for Generating Stepped-frequency RF Pulses**

** This appendix is presented in its original form without editorial change.

```
function [v, f_sample] = awg_stepped( f_start,f_end,f_delta,T_env,P_env,duty_cycle,f_sample_max )

f_res = 1/T_env;                            % frequency resolution (Hz)

f_sample = 2*f_res;
while (f_sample <= f_sample_max/2)          % adjust AWG sample rate
    f_sample = 2*f_sample;
end

freq = (f_start:f_delta:f_end-f_delta);     % instantanous frequencies
N = length(freq);

% adjust instantaneous frequencies
for counter = 1:length(freq)
    freq(counter) = floor(freq(counter)/f_res)*f_res;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

t_sbb = 1/f_sample;                         % sampling time
t = 0:t_sbb:T_env-t_sbb;                    % time vector
A = sqrt(10^(P_env/10)*2*50*10^-3);         % amplitude of chirp when ON

index = ceil((1:length(t))/(length(t)/N));
f = freq(index);
x = A * cos(2*pi*(f.*t));

if (duty_cycle <= 0) || (duty_cycle >= 1)
    v = x;                                  % continuous wave
else
    v = [x zeros(1,(round(1/duty_cycle)-1)*length(x))];     % pulsed
end
```

Appendix E. MATLAB Script for the Graphical User Interface††

†† This appendix is presented in its original form without editorial change.

```
function varargout = gremlin(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gremlin_OpeningFcn, ...
                   'gui_OutputFcn',  @gremlin_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% executes before the GUI is made visible %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function gremlin_OpeningFcn(hObject, eventdata, handles, varargin)

% connect to the DC power supplies (Agilent E3631A)
global dca dcb;
dca = visa('ni','GPIB0::5::INSTR');
dcb = visa('ni','GPIB0::4::INSTR');

fopen(dca);
fopen(dcb);
fprintf(dca, ['APPLy P6V, 5 V']);       % apply 5V to RF switches
fprintf(dcb, ['APPLy P6V, 3 V']);       % apply 3V to low-noise amplifiers
fprintf(dca, ['APPLy P25V, 0 V']);      % start up in linear Rx mode
fprintf(dcb, ['APPLy P25V, 5 V']);
fprintf(dca, ['OUTPut:STATe ON']);
fprintf(dcb, ['OUTPut:STATe ON']);
fclose(dca);
fclose(dcb);

% connect to the AWG (Tektronix AWG7052)
%  and to the oscilloscope (Lecroy Wavemaster 8300A)
global awg osc;
awg = visa('ni','GPIB0::2::INSTR');
osc = visa('ni','GPIB0::6::INSTR');

fopen(awg)
fwrite(awg, 'AWGCONTROL:DOUTPUT1:STATE 1');     % turn 'direct output' on
fclose(awg)

% define strings for the Tx channel (C2), Rx channel (C3),
%  Tx channel averaged (F2), and Rx channel averaged (F3)
global data_trc_1 data_trc_2 avg_trc_1 avg_trc_2 sweeps;
data_trc_1 = 'C2';
data_trc_2 = 'C3';
avg_trc_1 = 'F2';
avg_trc_2 = 'F3';
sweeps = 1;

% turn C2 and C3 off; turn F2 and F3 on
fopen(osc)
fprintf(osc,[avg_trc_1 ':DEF EQN,''AVG(' data_trc_1 '),SUMMED'',SWEEPS,' num2str(sweeps)]);
fprintf(osc,[avg_trc_2 ':DEF EQN,''AVG(' data_trc_2 '),SUMMED'',SWEEPS,' num2str(sweeps)]);
fprintf(osc,[data_trc_1 ':TRACE OFF']);
```

```
fprintf(osc,[data_trc_2 ':TRACE OFF']);
fprintf(osc,[avg_trc_1 ':TRACE ON']);
fprintf(osc,[avg_trc_2 ':TRACE ON']);
fprintf(osc,[avg_trc_1 ':FRST']);           % reset the averaged sweeps
fprintf(osc,[avg_trc_2 ':FRST']);
fclose(osc)

handles.output = hObject;
guidata(hObject, handles);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% generate single-tone RF pulse %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton1_Callback(hObject, eventdata, handles)

global awg;

% user-defined waveform name to be uploaded to the AWG
global wf_name;
wf_name = get(handles.edit6,'String');

% user-defined pulse power while the pulse is active
global P_env;
P_env = str2num(get(handles.edit1,'String'));

% user-defined carrier frequency for the RF pulse
global f_pulse;
f_pulse = str2num(get(handles.edit2,'String'));

% user-defined time for the RF pulse to be active
global T_env;
T_env = str2num(get(handles.edit4,'String'));

% user-defined duty cycle
global duty_cycle;
duty_cycle = str2num(get(handles.edit5,'String'))/100;

% user-defined maximum sampling rate for the AWG
global f_sample_max;
f_sample_max = str2num(get(handles.edit3,'String'));

% generate the single-tone pulse waveform
%  and compute an appropriate AWG sampling frequency
global v fsample;
[v,f_sample] = awg_pulse(f_pulse,T_env,P_env,duty_cycle,f_sample_max);

awg_plot(v,f_sample)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global samples;
samples = length(v);

% provide two markers (triggers) in sync with the AWG output
marker_1 = zeros(1,samples);
marker_1(1:round(samples/10)) = 1;
marker_2 = zeros(1,samples);
marker_2(1:round(samples/10)) = 1;

global buffer;
buffer = samples;
```

```
set(awg,'InputBufferSize',buffer);
set(awg,'OutputBufferSize',buffer);

% convert the real waveform data to binary
%  and upload the binary data to the AWG for playback
[binblock, header, bytes] = awg_binary(v, marker_1, marker_2);
awg_upload(awg, buffer, wf_name, binblock, header, bytes);

% set the AWG sampling frequency, select the waveform to be played,
%  and turn the AWG output on
fopen(awg);
fwrite(awg, 'AWGC:DOUT1 ON');
fwrite(awg, ['SOURCE1:FREQUENCY ' num2str(f_sample/10^9) ' GHZ']);
fwrite(awg, ['SOURCE1:WAVEFORM "' wf_name '"']);
fwrite(awg, 'OUTPUT ON');
fwrite(awg, 'AWGCONTROL:RUN');
fclose(awg);

global f_trans;
f_trans = f_pulse;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% generate linear RF chirp pulse %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton2_Callback(hObject, eventdata, handles)

global awg;

% user-defined waveform name to be uploaded to the AWG
global wf_name;
wf_name = get(handles.edit12,'String');

% user-defined chirp power while the pulse is active
global P_env;
P_env = str2num(get(handles.edit8,'String'));

% user-defined start frequency for the chirp
global f_start;
f_start = str2num(get(handles.edit9,'String'));

% user-defined end frequency for the chirp
global f_end;
f_end = str2num(get(handles.edit13,'String'));

% user-defined length of the chirp
global T_env;
T_env = str2num(get(handles.edit10,'String'));

% user-defined duty cycle
global duty_cycle;
duty_cycle = str2num(get(handles.edit11,'String'))/100;

% user-defined maximum sampling rate for the AWG
global f_sample_max;
f_sample_max = str2num(get(handles.edit3,'String'));

% generate the RF chirp waveform
%  and compute an appropriate AWG sampling frequency
global v f_sample;
[v,f_sample] = awg_chirp(f_start,f_end,T_env,P_env,duty_cycle,f_sample_max);
```

```
awg_plot(v,f_sample)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global samples;
samples = length(v);

% provide two markers (triggers) in sync with the AWG output
marker_1 = zeros(1,samples);
marker_1(1:round(samples/10)) = 1;
marker_2 = zeros(1,samples);
marker_2(1:round(samples/10)) = 1;

global buffer;
buffer = samples;
set(awg,'InputBufferSize',buffer);
set(awg,'OutputBufferSize',buffer);

% convert the real waveform data to binary
%  and upload the binary data to the AWG for playback
[binblock, header, bytes] = awg_binary(v, marker_1, marker_2);
awg_upload(awg, buffer, wf_name, binblock, header, bytes);

% set the AWG sampling frequency, select the waveform to be played,
%  and turn the AWG output on
fopen(awg);
fwrite(awg, 'AWGC:DOUT1 ON');
fwrite(awg, ['SOURCE1:FREQUENCY ' num2str(f_sample/10^9) ' GHZ']);
fwrite(awg, ['SOURCE1:WAVEFORM "' wf_name '"']);
fwrite(awg, 'OUTPUT ON');
fwrite(awg, 'AWGCONTROL:RUN');
fclose(awg);

global f_trans;
f_trans = (f_start + f_end)/2;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% generate multitone RF pulse %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton4_Callback(hObject, eventdata, handles)

global awg;

% user-defined waveform name to be uploaded to the AWG
global wf_name;
wf_name = get(handles.edit18,'String');

global N;
N = str2num(get(handles.edit19,'String'));

% user-defined power per tone while the pulse is active
global P_tone;
P_tone = str2num(get(handles.edit14,'String'));

% user-defined center frequency for the tones
global f_c;
f_c = str2num(get(handles.edit15,'String'));

% user-defined time for the multitone pulse to be active
global T_env;
T_env = str2num(get(handles.edit16,'String'));
```

```
% user-defined duty cycle
global duty_cycle;
duty_cycle = str2num(get(handles.edit17,'String'))/100;

% user-defined maximum sampling rate for the AWG
global f_sample_max;
f_sample_max = str2num(get(handles.edit3,'String'));

% generate the multitone waveform
%  and compute an appropriate AWG sampling frequency
global v fsample;
[v,f_sample] = awg_multitone(N,f_c,T_env,P_tone,duty_cycle,f_sample_max);

awg_plot(v,f_sample)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global samples;
samples = length(v);

% provide two markers (triggers) in sync with the AWG output
marker_1 = zeros(1,samples);
marker_1(1:round(samples/10)) = 1;
marker_2 = zeros(1,samples);
marker_2(1:round(samples/10)) = 1;

global buffer;
buffer = samples;
set(awg,'InputBufferSize',buffer);
set(awg,'OutputBufferSize',buffer);

% convert the real waveform data to binary
%  and upload the binary data to the AWG for playback
[binblock, header, bytes] = awg_binary(v, marker_1, marker_2);
awg_upload(awg, buffer, wf_name, binblock, header, bytes);

% set the AWG sampling frequency, select the waveform to be played,
%  and turn the AWG output on
fopen(awg);
fwrite(awg, 'AWGC:DOUT1 ON');
fwrite(awg, ['SOURCE1:FREQUENCY ' num2str(f_sample/10^9) ' GHZ']);
fwrite(awg, ['SOURCE1:WAVEFORM "' wf_name '"']);
fwrite(awg, 'OUTPUT ON');
fwrite(awg, 'AWGCONTROL:RUN');
fclose(awg);

global f_trans;
f_trans = f_c;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% generate stepped-frequency pulse %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton5_Callback(hObject, eventdata, handles)

global awg;

% user-defined waveform name to be uploaded to the AWG
global wf_name;
wf_name = get(handles.edit24,'String');
```

```
% user-defined power at each frequency while the pulse is active
global P_env;
P_env = str2num(get(handles.edit20,'String'));

% user-defined start frequency for the steps
global f_start;
f_start = str2num(get(handles.edit26,'String'));

% user-defined end frequency for the steps
global f_end;
f_end = str2num(get(handles.edit21,'String'));

% user-defined frequency step size
global f_delta;
f_delta = str2num(get(handles.edit25,'String'));

% user-defined time for the stepped waveform to be active
global T_env;
T_env = str2num(get(handles.edit22,'String'));

% user-defined duty cycle
global duty_cycle;
duty_cycle = str2num(get(handles.edit23,'String'))/100;

% user-defined maximum sampling rate for the AWG
global f_sample_max;
f_sample_max = str2num(get(handles.edit3,'String'));

% generate the stepped-frequency waveform
%  and compute an appropriate AWG sampling frequency
global v f_sample;
[v,f_sample] = awg_stepped(f_start,f_end,f_delta,T_env,P_env,duty_cycle,f_sample_max);


awg_plot(v,f_sample)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global samples;
samples = length(v);

% provide two markers (triggers) in sync with the AWG output
marker_1 = zeros(1,samples);
marker_1(1:round(samples/10)) = 1;
marker_2 = zeros(1,samples);
marker_2(1:round(samples/10)) = 1;

global buffer;
buffer = samples;
set(awg,'InputBufferSize',buffer);
set(awg,'OutputBufferSize',buffer);

% convert the real waveform data to binary
%  and upload the binary data to the AWG for playback
[binblock, header, bytes] = awg_binary(v, marker_1, marker_2);
awg_upload(awg, buffer, wf_name, binblock, header, bytes);

% set the AWG sampling frequency, select the waveform to be played,
%  and turn the AWG output on
fopen(awg);
fwrite(awg, 'AWGC:DOUT1 ON');
fwrite(awg, ['SOURCE1:FREQUENCY ' num2str(f_sample/10^9) ' GHZ']);
fwrite(awg, ['SOURCE1:WAVEFORM "' wf_name '"']);
fwrite(awg, 'OUTPUT ON');
fwrite(awg, 'AWGCONTROL:RUN');
```

```
fclose(awg);

global f_trans;
f_trans = (f_start + f_end)/2;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% clear waveforms & reset AWG %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton3_Callback(hObject, eventdata, handles)

global awg;

fopen(awg);
fwrite(awg,'*RST');
fwrite(awg,'*CLS');
pause(1)
fwrite(awg, 'AWGCONTROL:DOUTPUT1:STATE 1');
fclose(awg);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% capture and process Rx waveform %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function pushbutton7_Callback(hObject, eventdata, handles)

global mode;
switch get(handles.popupmenu3,'Value')
    case 1
        mode = 1;                   % select LINEAR receive mode
    case 2
        mode = 2;                   % select NONLINEAR receive mode
    otherwise
        mode = 1;
end

global dca dcb;
fopen(dca);
fopen(dcb);
if (mode == 2)
    fprintf(dca, ['APPLy P25V, 5 V']);      % set RF switches to
    fprintf(dcb, ['APPLy P25V, 0 V']);      %  nonlinear Rx chain
else
    fprintf(dca, ['APPLy P25V, 0 V']);      % set RF switches to
    fprintf(dcb, ['APPLy P25V, 5 V']);      %  linear Rx chain
end
fclose(dca);
fclose(dcb);

global awg osc;

global trig_src;
switch get(handles.popupmenu2,'Value')
    case 1
        trig_src = 'EX';                    % trigger = external
    case 2
        trig_src = 'C1';                    % trigger = channel C1
    case 3
        trig_src = 'C4';                    % trigger = channel C4
    otherwise
        trig_src = 'EX';
end
```

```
% user-defined trigger level
global trig_lev;
trig_lev = str2num(get(handles.edit37,'String'))*10^-3;

% user-defined total time for Tx and Rx data to be collected
global t_total;
t_total = str2num(get(handles.edit32,'String'))*10^-6;
t_div = t_total/10;

% user-defined voltage per division visible on the oscilloscope
global v_div_1 v_div_2;
v_div_1 = str2num(get(handles.edit33,'String'))*10^-3;
v_div_2 = str2num(get(handles.edit34,'String'))*10^-3;

% user-defined voltage offset on the oscilloscope
global v_offset;
v_offset = 0e-3;

% set trigger delay to 1/2 of the data collection time so that
%  the maximum amount of data is collected AFTER the trigger point
global trig_delay;
trig_delay = -t_total/2;

% user-defined Matlab file to store Tx and Rx time & voltage vectors
global data_name;
data_name = get(handles.edit36,'String');

global trig_mode;
trig_mode = 'SINGLE';           % use a single trigger for each sweep

global f_capture;
f_capture = 20e9;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global buffer;
buffer = 10*t_div*f_capture + 26;
set(osc,'InputBufferSize',buffer);
set(osc,'OutputBufferSize',buffer);

global data_trc_1 data_trc_2 avg_trc_1 avg_trc_2 sweeps;
sweeps = str2num(get(handles.edit38,'String'));

% set the oscilloscope to record data according to user-input values
fopen(osc);
fprintf(osc, ['TRSE EDGE,SR,' trig_src ',HT,OFF']);
fprintf(osc, ['TRMD ' trig_mode]);
fprintf(osc, ['TRDL ' num2str(trig_delay/10^-6) ' US']);
fprintf(osc, ['EX:TRLV ' num2str(trig_lev/10^-3) ' MV']);
fprintf(osc, ['TDIV ' num2str(t_div/10^-6) ' US']);
fprintf(osc, [data_trc_1 ':VDIV ' num2str(v_div_1/10^-3) ' MV']);
fprintf(osc, [data_trc_2 ':VDIV ' num2str(v_div_2/10^-3) ' MV']);
fprintf(osc, [data_trc_1 ':OFST ' num2str(v_offset/10^-3) ' MV']);
fprintf(osc, [data_trc_2 ':OFST ' num2str(v_offset/10^-3) ' MV']);
fclose(osc);

% turn the appropriate data traces on/off
%  and set two of the traces to each average the Tx or Rx data stream
fopen(osc);
fprintf(osc,[data_trc_1 ':TRACE OFF']);
fprintf(osc,[data_trc_2 ':TRACE OFF']);
fprintf(osc,[avg_trc_1 ':TRACE ON']);
fprintf(osc,[avg_trc_2 ':TRACE ON']);
fprintf(osc,[avg_trc_1 ':FRST']);
```

```
fprintf(osc,[avg_trc_2 ':FRST']);
fprintf(osc,[avg_trc_1 ':DEF EQN,''AVG(' data_trc_1 '),SUMMED'',SWEEPS,' num2str(sweeps)]);
fprintf(osc,[avg_trc_2 ':DEF EQN,''AVG(' data_trc_2 '),SUMMED'',SWEEPS,' num2str(sweeps)]);
for counter = 1:sweeps
    fprintf(osc,'*TRG');
    fprintf(osc,'WAIT');
end
fclose(osc);

% download the oscilloscope data into Matlab
[v_trans, v_rec, t_trans, t_rec] = osc_capture( osc, buffer, avg_trc_1, avg_trc_2 );

% stop the waveform generator playback
fopen(awg);
fwrite(awg, 'OUTPUT OFF');
fwrite(awg, 'AWGCONTROL:STOP');
fclose(awg);

global f_trans;

% process and display the Tx and Rx data
osc_process( v_trans, v_rec, t_trans, t_rec, mode, f_trans );

% save the raw Tx and Rx vectors to a MAT file
save(data_name,'t_trans','v_trans','t_rec','v_rec');



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% initialize all text boxes, buttons, etc. %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function varargout = gremlin_OutputFcn(hObject, eventdata, handles)
varargout{1} = handles.output;

function edit1_Callback(hObject, eventdata, handles)

function edit1_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit2_Callback(hObject, eventdata, handles)

function edit2_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit3_Callback(hObject, eventdata, handles)

function edit3_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit4_Callback(hObject, eventdata, handles)
```

```
function edit4_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit5_Callback(hObject, eventdata, handles)

function edit5_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit6_Callback(hObject, eventdata, handles)

function edit6_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit8_Callback(hObject, eventdata, handles)

function edit8_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit9_Callback(hObject, eventdata, handles)

function edit9_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit10_Callback(hObject, eventdata, handles)

function edit10_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit11_Callback(hObject, eventdata, handles)

function edit11_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit12_Callback(hObject, eventdata, handles)

function edit12_CreateFcn(hObject, eventdata, handles)
```

```
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit13_Callback(hObject, eventdata, handles)

function edit13_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit14_Callback(hObject, eventdata, handles)

function edit14_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit15_Callback(hObject, eventdata, handles)

function edit15_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit16_Callback(hObject, eventdata, handles)

function edit16_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit17_Callback(hObject, eventdata, handles)

function edit17_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit18_Callback(hObject, eventdata, handles)

function edit18_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit19_Callback(hObject, eventdata, handles)

function edit19_CreateFcn(hObject, eventdata, handles)
```

```
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit20_Callback(hObject, eventdata, handles)

function edit20_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit21_Callback(hObject, eventdata, handles)

function edit21_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit22_Callback(hObject, eventdata, handles)

function edit22_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit23_Callback(hObject, eventdata, handles)

function edit23_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit24_Callback(hObject, eventdata, handles)

function edit24_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit25_Callback(hObject, eventdata, handles)

function edit25_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit26_Callback(hObject, eventdata, handles)

function edit26_CreateFcn(hObject, eventdata, handles)
```

```
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function edit32_Callback(hObject, eventdata, handles)

function edit32_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function edit33_Callback(hObject, eventdata, handles)

function edit33_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function edit34_Callback(hObject, eventdata, handles)

function edit34_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function edit36_Callback(hObject, eventdata, handles)

function edit36_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function edit37_Callback(hObject, eventdata, handles)

function edit37_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function popupmenu2_Callback(hObject, eventdata, handles)

function popupmenu2_CreateFcn(hObject, eventdata, handles)

if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function popupmenu3_Callback(hObject, eventdata, handles)

function popupmenu3_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit38_Callback(hObject, eventdata, handles)

function edit38_CreateFcn(hObject, eventdata, handles)
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

INTENTIONALLY LEFT BLANK.

Appendix F. MATLAB Function for an Ideal Bandpass Filter‡‡

‡‡ This appendix is presented in its original form without editorial change.

```
function y_out = bandpass( x_in, y_in, start_freq, end_freq )

% y_out = bandpass( x_in, y_in, start_freq, end_freq )
%  -- applies an ideal bandpass filter to 'y_in' as a function of 'x_in'
%  -- start_freq = lower bandpass corner frequency
%  -- end_freq = upper bandpass corner frequency
%  -- y_out = filtered result, still a function of 'x_in'

T = x_in(length(x_in)) - x_in(1);              % full period of x

tau = x_in(2) - x_in(1);                       % delta(x) sample length

Y_temp = fft(y_in);                            % Fast Fourier Transform

if mod(length(Y_temp),2) == 1
    frequency = [ (0:floor(length(Y_temp)/2))*(1/T)  fliplr((1:floor(length(Y_temp)/2))*(-1/T))
];
else
    frequency = [ (0:length(Y_temp)/2)*(1/T)  fliplr((1:length(Y_temp)/2-1)*(-1/T)) ];
end

for n = 1:length(frequency)
    if ( frequency(n) >= -1*start_freq ) && ( frequency(n) <= start_freq ) ...
        || ( frequency(n) <= -1*end_freq ) || ( frequency(n) >= end_freq )
            Y_temp(n) = 0;
    end
end

y_out = real(ifft(Y_temp));                    % inverse FFT
```

Appendix G. MATLAB Function for Converting Real AWG Waveforms to Binary Data[§§]

[§§] This appendix is presented in its original form without editorial change.

```
function [binblock, header, bytes] = awg_binary(v, marker_1, marker_2)

y = 2 * single(v);

samples = length(v);

marker_1 = uint8(marker_1);
marker_2 = uint8(marker_2);

marker_1 = bitshift(uint8(logical(marker_1)),6);
marker_2 = bitshift(uint8(logical(marker_2)),7);
m = marker_1 + marker_2;

binblock = zeros(1,samples*5,'uint8');
for k=1:samples
    binblock((k-1)*5+1:(k-1)*5+5) = [typecast(y(k),'uint8') m(k)];
end

bytes = num2str(length(binblock));

header = ['#' num2str(length(bytes)) bytes];
```

Appendix H. MATLAB Function for Uploading Waveform Data to the AWG[***]

[***] This appendix is presented in its original form without editorial change.

```
function awg_upload( awg, buffer, wf_name, binblock, header, bytes)

fopen(awg);                     % connect to the waveform generator

fwrite(awg,['WLISt:WAVeform:DELete "' wf_name '"']);
fwrite(awg,['WLIST:WAVEFORM:NEW "' wf_name '",' num2str(buffer) ',REAL']);

cmd = ['WLIST:WAVEFORM:DATA "' wf_name '",' header binblock];

bytes = length(cmd);
if buffer >= bytes
    fwrite(awg,cmd)
else
    awg.EOIMode = 'off';
    for i = 1:buffer:bytes-buffer
        fwrite(awg,cmd(i:i+buffer-1))
    end
    awg.EOIMode = 'on';
    i = i + buffer;
    fwrite(awg,cmd(i:end))
end

fclose(awg);
```

Appendix I. MATLAB Function for Plotting Waveforms Uploaded to the AWG†††

††† This appendix is presented in its original form without editorial change.

```
function awg_plot( v, f_sample )

t = (0:length(v)-1)*1/f_sample;             % time vector
T = t(length(t)) + 1/f_sample;              % total time per waveform
f = -f_sample/2:1/T:(f_sample/2)-1/T;       % frequency grid

V = fftshift(fft(2*(v))/length(v));         % calculate FFT
PdB = 10*log10(abs(V).^2/(2*50*10^-3));     % calculate power spectrum

figure(1)                                   % plot in time & frequency
subplot(1,2,1)
plot(t/10^-6,v/10^-3)
axis([-Inf Inf 1.1*min(v/10^-3) 1.1*max(v/10^-3)])
ylabel('Voltage  (mV)')
xlabel('Time  (\mus)')
subplot(1,2,2)
plot(f/10^6,PdB)
axis([0 Inf -Inf max(PdB)+10])
ylabel('Power  (dBm)')
xlabel('Frequency  (MHz)')
```

Appendix J. MATLAB Function for Recording Real Waveforms Using the Oscilloscope‡‡‡

‡‡‡ This appendix is presented in its original form without editorial change.

```
function [v_1, v_2, t_1, t_2] = osc_capture( osc, buffer, data_trc_1, data_trc_2 )

fopen(osc);                                 % connect to the oscilloscope
fprintf(osc, 'ARM')


pause(1)
fprintf(osc, [data_trc_1 ':WF? ALL']);      % grab raw data trace
raw_data = fread(osc,buffer,'uint8');

% calculate voltage amplitude per step
v_quant_temp = raw_data(181)*16^6 + raw_data(180)*16^4 + raw_data(179)*16^2 + raw_data(178)*16^0;
v_quant = typecast(uint32(v_quant_temp),'single');


% calculate voltage offset
v_offset_temp = raw_data(185)*16^6 + raw_data(184)*16^4 + raw_data(183)*16^2 +
raw_data(182)*16^0;
v_offset = typecast(uint32(v_offset_temp),'single');


% calculate time interval per step
t_quant_temp = raw_data(201)*16^6 + raw_data(200)*16^4 + raw_data(199)*16^2 + raw_data(198)*16^0;
t_quant = typecast(uint32(t_quant_temp),'single');


% calculate time offset
t_offset_temp = raw_data(209)*16^14 + raw_data(208)*16^12 + raw_data(207)*16^10 + ...
raw_data(206)*16^8 raw_data(205)*16^6 + raw_data(204)*16^4 + raw_data(203)*16^2 + ...
raw_data(202)*16^0;


fprintf(osc, [data_trc_1 ':WF? DAT1']);     % grab voltage trace
raw_data_v = fread(osc,buffer,'int8');


% convert raw data to time & voltage waveforms
v_1 = v_quant*raw_data_v(23:length(raw_data_v)-1) + v_offset;
t_1 = t_quant*(0:length(v_1)-1) + t_offset;


pause(1)
fprintf(osc, [data_trc_2 ':WF? ALL']);      % grab raw data trace
raw_data = fread(osc,buffer,'uint8');


% calculate voltage amplitude per step
v_quant_temp = raw_data(181)*16^6 + raw_data(180)*16^4 + raw_data(179)*16^2 + raw_data(178)*16^0;
v_quant = typecast(uint32(v_quant_temp),'single');


% calculate voltage offset
v_offset_temp = raw_data(185)*16^6 + raw_data(184)*16^4 + raw_data(183)*16^2 +
raw_data(182)*16^0;
v_offset = typecast(uint32(v_offset_temp),'single');


% calculate time interval per step
t_quant_temp = raw_data(201)*16^6 + raw_data(200)*16^4 + raw_data(199)*16^2 + raw_data(198)*16^0;
t_quant = typecast(uint32(t_quant_temp),'single');


% calculate time offset
t_offset_temp = raw_data(209)*16^14 + raw_data(208)*16^12 + raw_data(207)*16^10 + ...
raw_data(206)*16^8 raw_data(205)*16^6 + raw_data(204)*16^4 + raw_data(203)*16^2 + ...
raw_data(202)*16^0;
t_offset = typecast(uint64(t_offset_temp),'double');


fprintf(osc, [data_trc_2 ':WF? DAT1']);     % grab voltage trace
raw_data_v = fread(osc,buffer,'int8');


% convert raw data to time & voltage waveforms
v_2 = v_quant*raw_data_v(23:length(raw_data_v)-1) + v_offset;
t_2 = t_quant*(0:length(v_2)-1) + t_offset;


fclose(osc);
```

Appendix K. MATLAB Function for Processing Radar Data from the Oscilloscope§§§

§§§ This appendix is presented in its original form without editorial change.

```
function osc_process( v_1, v_2, t_1, t_2, mode, f_trans )

delta_t = t_2(2) - t_2(1);
T = t_1(length(t_1)) - t_1(1);

% for nonlinear mode, correlate Rx with 2nd harmonic of Tx
if ( mode == 2 )
    v_1x = v_1.^2;
    v_1x = bandpass(t_1,v_1x,1.5*f_trans,2.5*f_trans);
else
    v_1x = v_1;
end

% compute correlation and associated time vector
v_corr = xcorr(v_1x,v_2);
t_corr = delta_t*(0:length(v_corr)-1) - T;

% adjust the velocity of propagation by dielectric constant
dielectric_PTFE = 2.1;
velocity_PTFE = 0.9836e9 / sqrt(dielectric_PTFE);

% compute a distance vector based on 2X transit time
d_corr = -1/2 * t_corr * velocity_PTFE;

% find the peak of the correlation waveform
[v_max,index] = max(abs(v_corr));
v_max = abs(v_corr(index));
d_max = d_corr(index);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(2)                               % plot the raw Tx and Rx data
subplot(2,1,1)
plot(t_1/10^-6,v_1,'-')
axis([-Inf Inf -1.5 1.5])
ylabel('Tx Voltage, Ch2  (V)')
subplot(2,1,2)
plot(t_2/10^-6,v_2/10^-3,'-')
axis([-Inf Inf -250 250])
ylabel('Rx Voltage, Ch 3  (mV)')
xlabel('Time  (\mus)')

figure(3)                               % plot Tx and Rx data samples
subplot(2,1,1)                          %  to confirm Rx frequency
plot(t_1/10^-9,v_1,'-')                 %  is twice the Tx frequency
axis([500 502 -1.5 1.5])
ylabel('Tx Voltage, Ch2  (V)')
grid
subplot(2,1,2)
plot(t_2/10^-9,v_2/10^-3,'-')
axis([500 502 -150 150])
ylabel('Rx Voltage, Ch 3  (mV)')
xlabel('Time  (ns)')
grid

figure(5)                               % plot the Tx and Rx correlation
plot(d_corr,v_corr)
axis([50 400 -1000 1000])
% text(250,800,['peak @ {\itd} = ' num2str(round(d_max)) ' ft']);
ylabel('{\bfCorrelation,  {\itV}_{trans} * {\itV}_{rec}}')
xlabel('{\bfDistance to Target}  (ft)')
```

List of Symbols, Abbreviations, and Acronyms

ARL	U.S. Army Research Laboratory
AWG	arbitrary waveform generator
COTS	commercial off-the-shelf
CW	continuous-wave
FM	frequency modulated
FRS	Family Radio Service
GPIB	General Purpose Interface Bus
GUI	graphical user interface
RF	radio frequency
Rx	receiver
SIRE	Synchronous Impulse Reconstruction
SMA	Subminiature Version A
Tx	transmitter

NO. OF COPIES	ORGANIZATION
1 (PDF)	1 DEFENSE TECHNICAL INFORMATION CTR DTIC OCA 8725 JOHN J KINGMAN RD STE 0944 FORT BELVOIR VA 22060-6218
1 (HC)	DIRECTOR US ARMY RESEARCH LAB IMAL HRA 2800 POWDER MILL RD ADELPHI MD 20783-1197
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL 2800 POWDER MILL RD ADELPHI MD 20783-1197
1 (PDF)	GOVT PRINTG OFC A MALHOTRA 732 N CAPITOL ST NW WASHINGTON DC 20401
13 PDFS	DIRECTOR US ARMY RESEARCH LAB ATTN RDRL SER M A HEDDEN ATTN RDRL SER U T DOGARU M HIGGINS G KIROSE F KOENIG D LIAO A MARTONE D MCNAMARA G MAZZARO M RESSLER K SHERBONDY G SMITH A SULLIVAN 2800 POWDER MILL RD ADELPHI MD 20783-1197

TOTAL: 17 (16 ELEC, 1 HC)